

Automatic Performance Tuning of Numerical Kernels

BeBOP: Berkeley Benchmarking and OPTimization

James Demmel
EECS and Math
UC Berkeley

Katherine Yelick
EECS
UC Berkeley

Support from NSF, DOE SciDAC, Intel

Performance Tuning Participants

Other Faculty

 Michael Jordan, William Kahan, Zhaojun Bai (UCD)

Researchers

 Mark Adams (SNL), David Bailey (LBL), Parry Husbands (LBL), Xiaoye Li (LBL), Lenny Oliker (LBL)

PhD Students

 **Rich Vuduc**, Yozo Hida, Geoff Pike

Undergrads

 Brian Gaeke , Jen Hsu, Shoaib Kamil, Suh Kang, Hyun Kim, Gina Lee, Jaeseop Lee, Michael de Lorimier, Jin Moon, Randy Shoopman, Brandon Thompson

Outline

- ~~✍~~ Motivation, History, Related work
 - ~~✍~~ Tuning Sparse Matrix Operations
 - ~~✍~~ Results on Sun Ultra 1/170
 - ~~✍~~ Recent results on P4
 - ~~✍~~ Recent results on Itanium
 - ~~✍~~ Future Work
-

Motivation
History
Related Work

Performance Tuning

✍ Motivation: performance of many applications dominated by a few kernels

✍ CAD ✍ Nonlinear ODEs ✍ Nonlinear equations ✍ Linear equations ✍ Matrix multiply

✍ Matrix-by-matrix or matrix-by-vector

✍ Dense or Sparse

✍ Information retrieval by LSI ✍ Compress term-document matrix ✍ ... ✍ Sparse mat-vec multiply

✍ Many other examples (not all linear algebra)

Conventional Performance Tuning

- ✍ Motivation: performance of many applications dominated by a few kernels
 - ✍ Vendor or user hand tunes kernels
 - ✍ Drawbacks:
 - ✍ Very time consuming and tedious work
 - ✍ Even with intimate knowledge of architecture and compiler, performance hard to predict
 - ✍ Growing list of kernels to tune
 - ✍ Example: New BLAS Standard
 - ✍ Must be redone for every architecture, compiler
 - ✍ Compiler technology often lags architecture
 - ✍ Not just a compiler problem:
 - ✍ Best algorithm may depend on input, so some tuning at run-time.
 - ✍ Not all algorithms semantically or mathematically equivalent
-

Automatic Performance Tuning

✍ Approach: for each kernel

1. Identify and generate a space of algorithms
2. Search for the fastest one, by running them

✍ What is a space of algorithms?

✍ Depending on kernel and input, may vary

- ✍ instruction mix and order
- ✍ memory access patterns
- ✍ data structures
- ✍ mathematical formulation

✍ When do we search?

- ✍ Once per kernel and architecture
 - ✍ At compile time
 - ✍ At run time
 - ✍ All of the above
-

Some Automatic Tuning Projects

- ✍ PHIPAC (www.icsi.berkeley.edu/~bilmes/hipac) (Bilmes, Asanovic, Vuduc, Demmel)
 - ✍ ATLAS (www.netlib.org/atlas) (Dongarra, Whaley; in Matlab)
 - ✍ XBLAS (www.nersc.gov/~xiaoye/XBLAS) (Demmel, X. Li)
 - ✍ Sparsity (www.cs.berkeley.edu/~yelick/sparsity) (Yelick, Im)
 - ✍ FFTs and Signal Processing
 - ✍ FFTW (www.fftw.org)
 - ✍ Won 1999 Wilkinson Prize for Numerical Software
 - ✍ SPIRAL (www.ece.cmu.edu/~spiral)
 - ✍ Extensions to other transforms, DSPs
 - ✍ UHFFT
 - ✍ Extensions to higher dimension, parallelism
 - ✍ Special session at ICCS 2001
 - ✍ Organized by Yelick and Demmel
 - ✍ www.ucalgary.ca/iccs
 - ✍ Proceedings available
 - ✍ Pointers to other automatic tuning projects at
 - ✍ www.cs.berkeley.edu/~yelick/iccs-tune
-

PHIPAC: Removing False Dependencies

✍ Using local variables, reorder operations to remove false dependencies

```
a[i] = b[i] + c;  
a[i+1] = b[i+1] * d;
```

false read-after-write hazard
between a[i] and b[i+1]



```
float f1 = b[i];  
float f2 = b[i+1];
```

```
a[i] = f1 + c;  
a[i+1] = f2 * d;
```

✍ With some compilers, you can say explicitly (via flag or pragma) that a and b are not aliased.

PHIPAC: Exploiting Multiple Registers

✍️ Reduce demands on memory bandwidth by pre-loading into local variables

Before

```
while( ... ) {  
    *res++ = filter[0]*signal[0]  
            + filter[1]*signal[1]  
            + filter[2]*signal[2];  
    signal++;  
}
```



After

```
float f0 = filter[0];  
float f1 = filter[1];  
float f2 = filter[2];  
while( ... ) {  
    *res++ = f0*signal[0]  
            + f1*signal[1]  
            + f2*signal[2];  
    signal++;  
}
```

also: register float f0 = ...;

PHIPAC: Minimizing Pointer Updates

✍️ Replace pointer updates for strided memory addressing with constant array offsets

```
f0 = *r8; r8 += 4;  
f1 = *r8; r8 += 4;  
f2 = *r8; r8 += 4;
```



```
f0 = r8[0];  
f1 = r8[4];  
f2 = r8[8];  
r8 += 12;
```

PHIPAC: Loop Unrolling

✍️ Expose instruction-level parallelism

```
float f0 = filter[0], f1 = filter[1], f2 = filter[2];
float s0 = signal[0], s1 = signal[1], s2 = signal[2];
*res++ = f0*s0 + f1*s1 + f2*s2;
do {
    signal += 3;
    s0 = signal[0];
    res[0] = f0*s1 + f1*s2 + f2*s0;

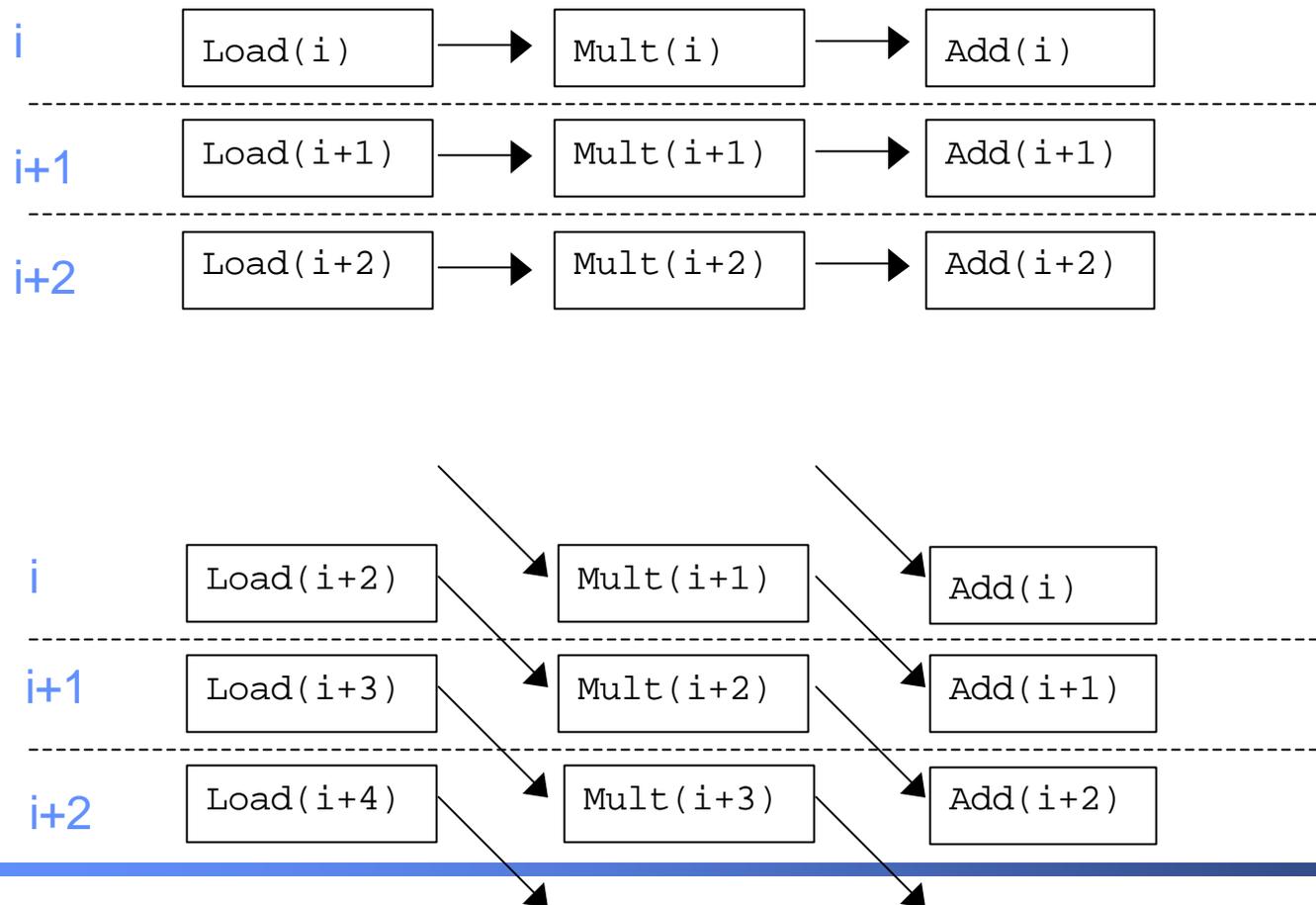
    s1 = signal[1];
    res[1] = f0*s2 + f1*s0 + f2*s1;

    s2 = signal[2];
    res[2] = f0*s0 + f1*s1 + f2*s2;

    res += 3;
} while( ... );
```

PHIPAC: Software Pipelining of Loops

- ✍ Mix instructions from different iterations
- ✍ Keeps functional units "busy"
- ✍ Reduce stalls from dependent instructions



PHIPAC: Exposing Independent Operations

✍ Hide instruction latency

- ✍ Use local variables to expose independent operations that can execute in parallel or in a pipelined fashion
- ✍ Balance the instruction mix (what functional units are available?)

```
f1 = f5 * f9;  
f2 = f6 + f10;  
f3 = f7 * f11;  
f4 = f8 + f12;
```

PHIPAC: Copy optimization

- ✍ Copy input operands
 - ✍ Reduce cache conflicts
 - ✍ Constant array offsets for fixed size blocks
 - ✍ Expose page-level locality

Original matrix, column major
(**numbers are addresses**)

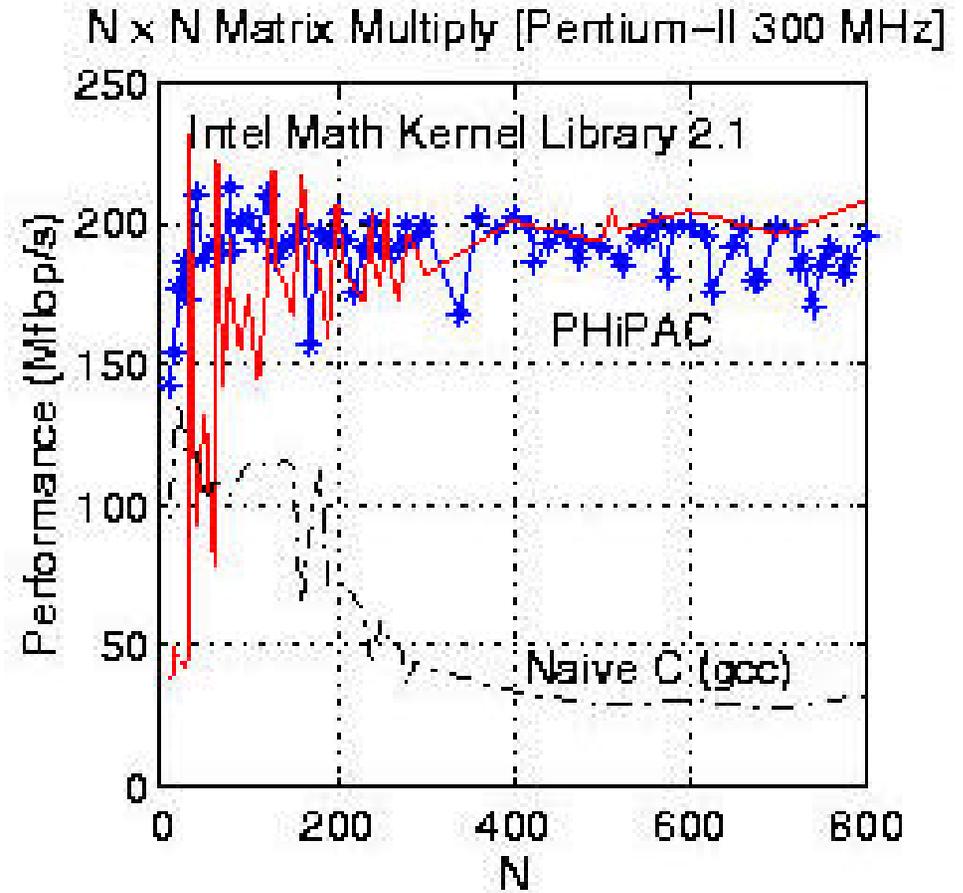
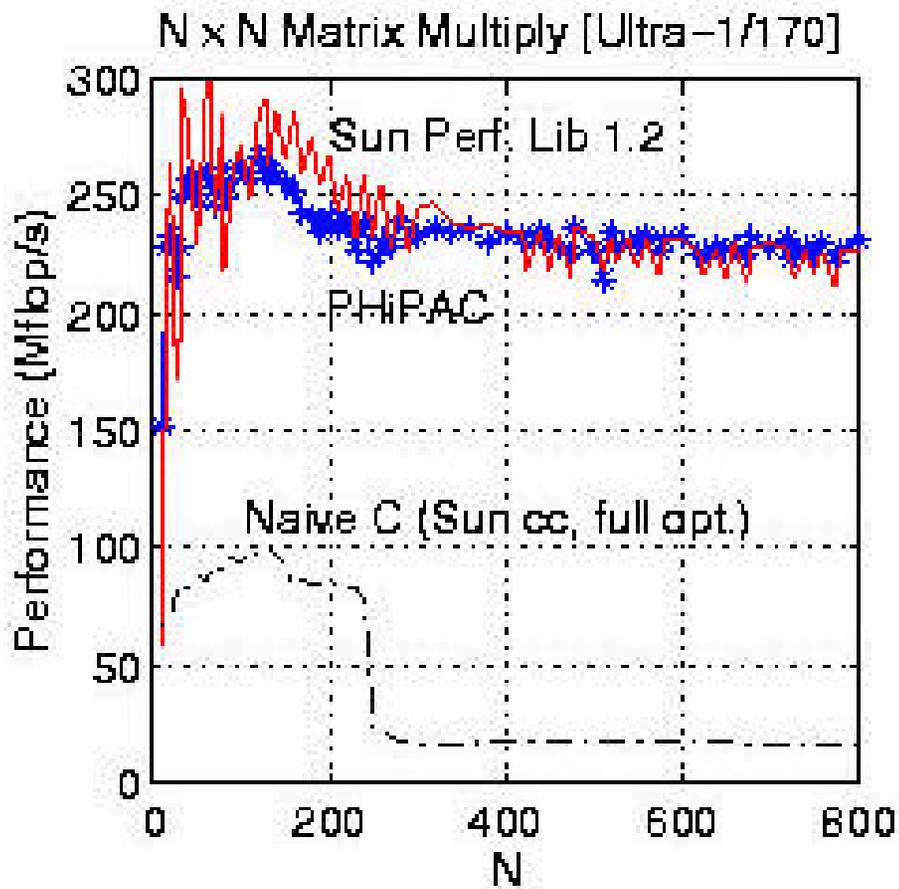


0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

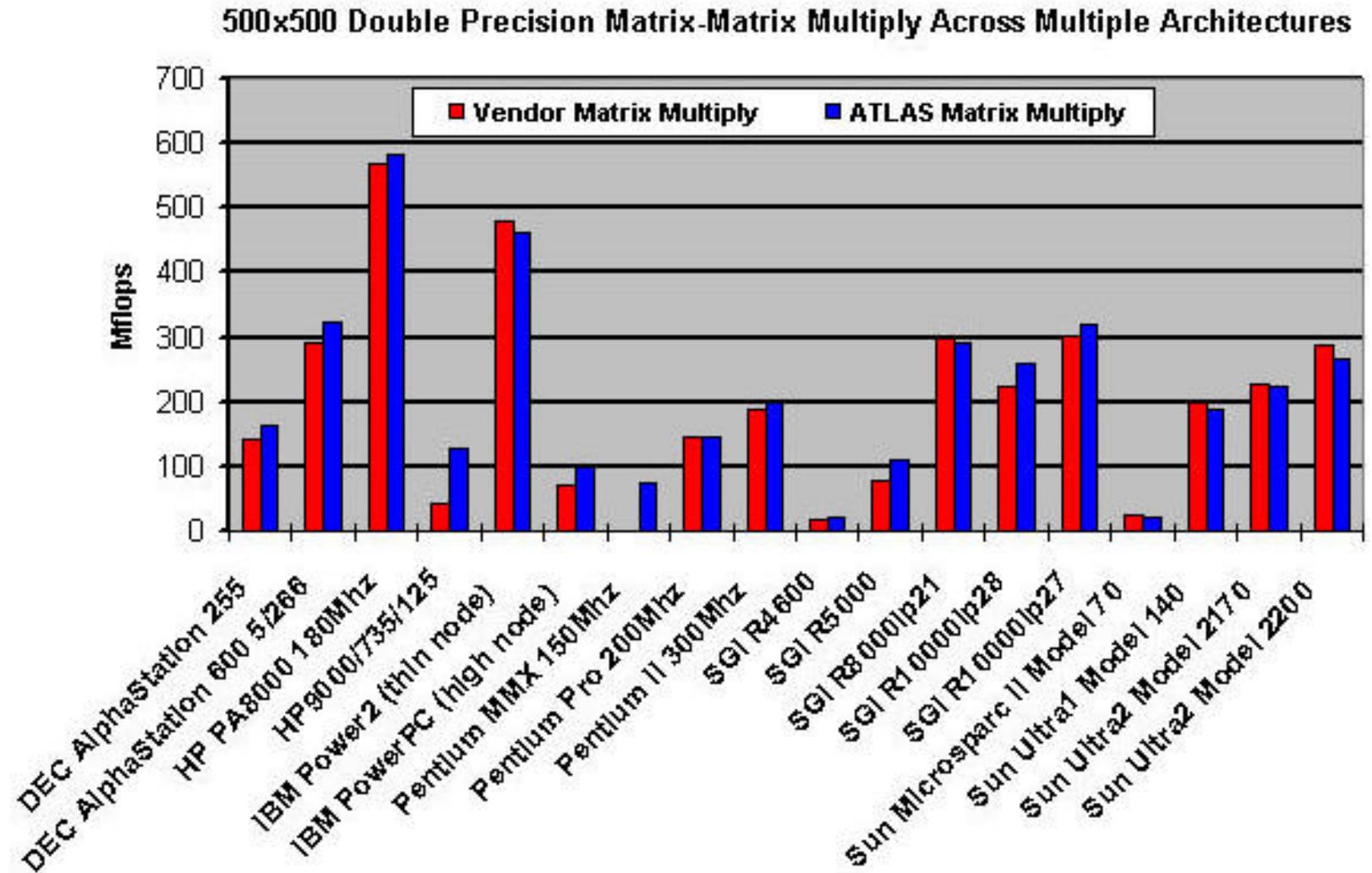
Reorganized into
2x2 blocks

0	2	8	10
1	3	9	11
4	6	12	13
5	7	14	15

Tuning pays off - PHIPAC (Bilmes, Asanovic, Demmel)

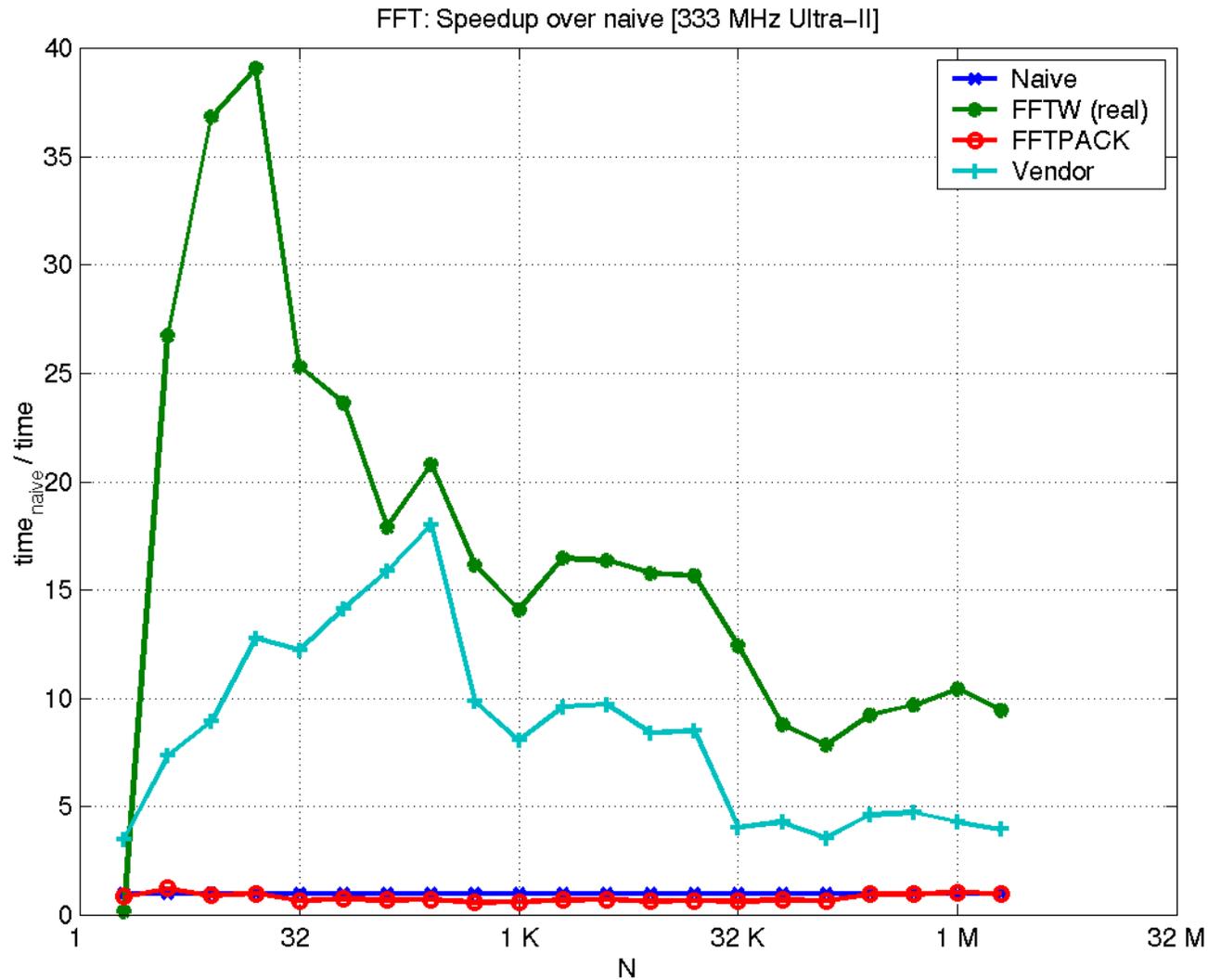


Tuning pays off - ATLAS (Dongarra, Whaley)



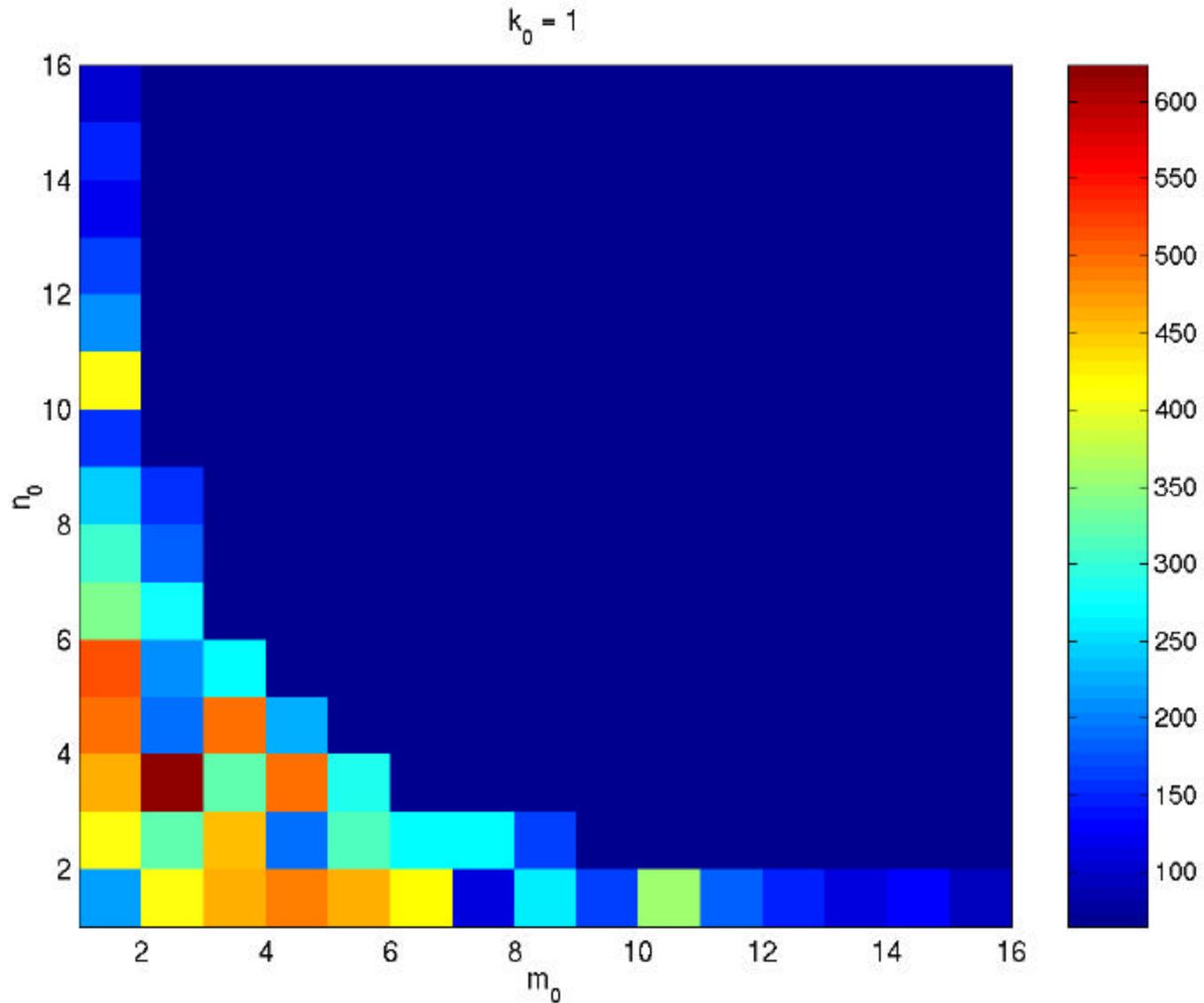
Extends applicability of PHIPAC
Incorporated in Matlab (with rest of LAPACK)

Tuning pays off - FFTW (Frigo, Johnson)



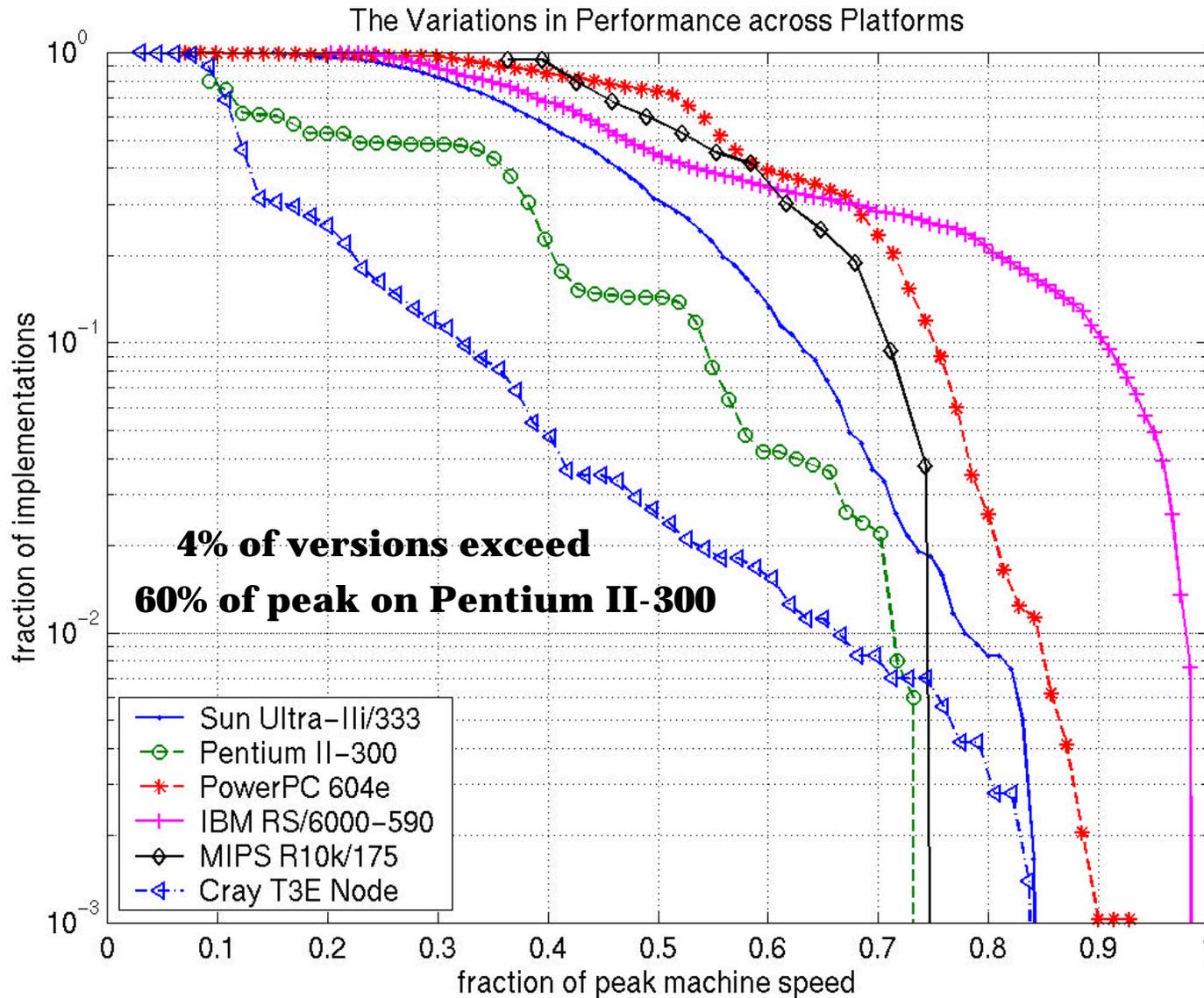
**Tuning ideas applied to signal processing
(DFT) Also incorporated in Matlab**

Search for optimal register tile sizes on Sun Ultra 10

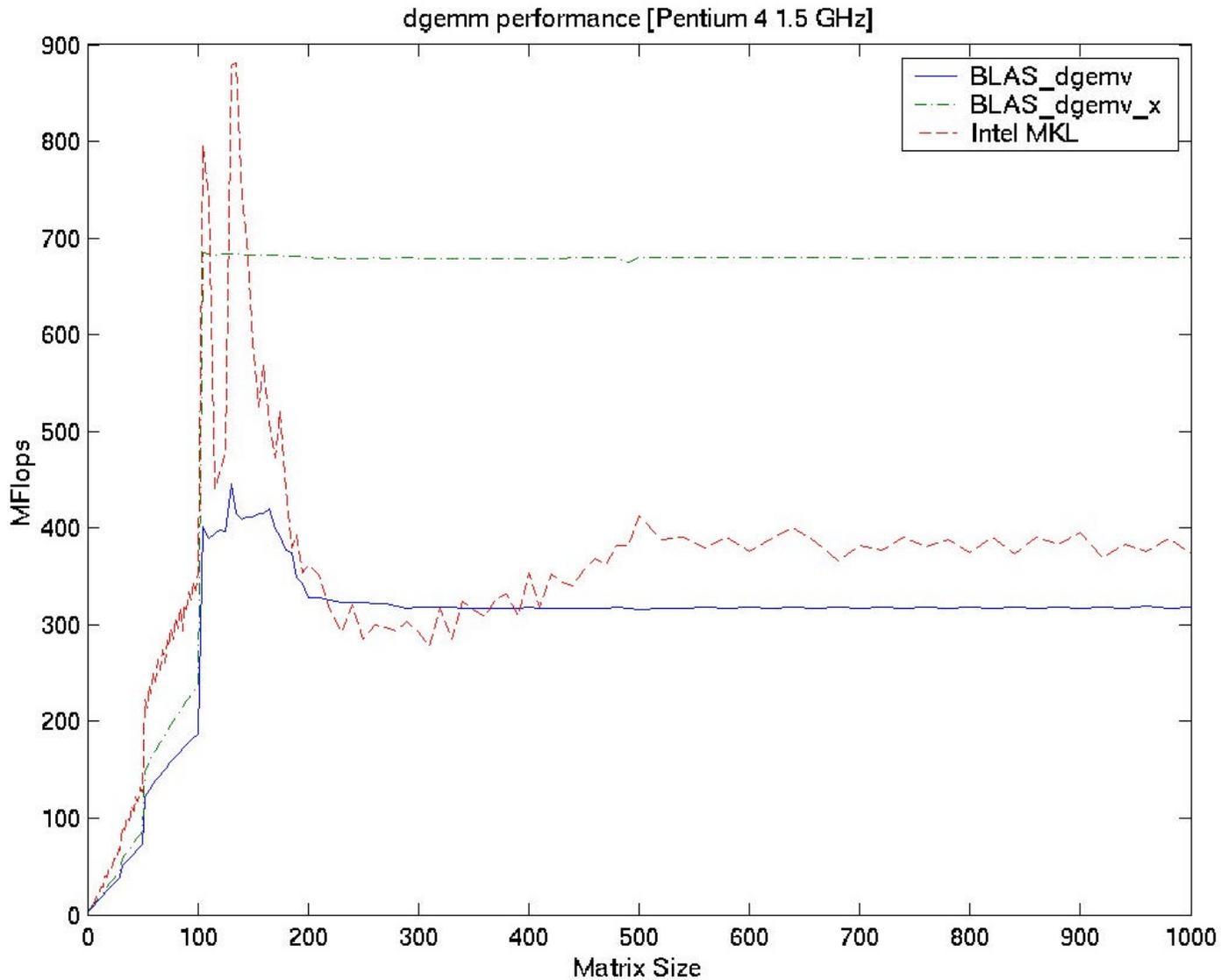


16 registers, but 2-by-3 tile size fastest

Search for Optimal LO block size in dense matmul



High precision dense mat-vec multiply (XBLAS)



High Precision Algorithms (XBLAS)

- ✍ Double-double (High precision word represented as pair of doubles)
 - ✍ Many variations on these algorithms; we currently use Bailey's
 - ✍ Exploiting Extra-wide Registers
 - ✍ Suppose $s(1), \dots, s(n)$ have f -bit fractions, SUM has $F > f$ bit fraction
 - ✍ Consider following algorithm for $S = \sum_{i=1, n} s(i)$
 - ✍ Sort so that $|s(1)| \geq |s(2)| \geq \dots \geq |s(n)|$
 - ✍ SUM = 0, for $i = 1$ to n SUM = SUM + $s(i)$, end for, sum = SUM
 - ✍ Theorem (D., Hida) Suppose $F < 2f$ (less than double precision)
 - ✍ If $n \leq 2^{F-f} + 1$, then error ≤ 1.5 ulps
 - ✍ If $n = 2^{F-f} + 2$, then error $\leq 2^{2f-F}$ ulps (can be ≤ 1)
 - ✍ If $n \geq 2^{F-f} + 3$, then error can be arbitrary ($S \neq 0$ but sum = 0)
 - ✍ Examples
 - ✍ $s(i)$ double ($f=53$), SUM double extended ($F=64$)
 - accurate if $n \leq 2^{11} + 1 = 2049$
 - ✍ Dot product of single precision $x(i)$ and $y(i)$
 - $s(i) = x(i) * y(i)$ ($f=2*24=48$), SUM double extended ($F=64$) ?
 - accurate if $n \leq 2^{16} + 1 = 65537$
-

Tuning Sparse Matrix Computations

Tuning Sparse matrix-vector multiply

✍ Sparsity

- ✍ Optimizes $y = A * x$ for a particular sparse A

✍ Im and Yelick

✍ Algorithm space

- ✍ Different code organization, instruction mixes
- ✍ Different register blockings (change data structure and fill of A)
- ✍ Different cache blocking
- ✍ Different number of columns of x
- ✍ Different matrix orderings

✍ Software and papers available

- ✍ www.cs.berkeley.edu/~yelick/sparsity
-

How Sparsity tunes $y = A*x$

✍ Register Blocking

- ✍ Store matrix as dense $r \times c$ blocks
- ✍ Precompute performance in **Mflops** of dense $A*x$ for various register block sizes $r \times c$
- ✍ Given A , sample it to estimate **Fill** if A blocked for varying $r \times c$
- ✍ Choose $r \times c$ to minimize estimated running time **Fill/Mflops**
 - ✍ Store explicit zeros in dense $r \times c$ blocks, unroll

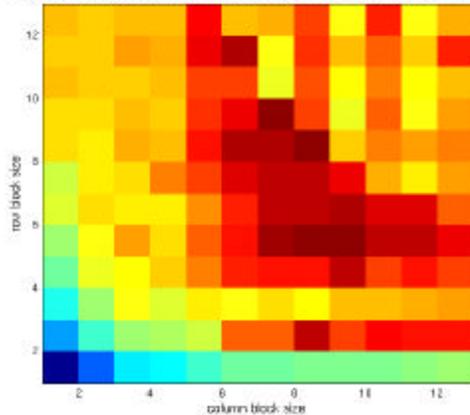
✍ Cache Blocking

- ✍ Useful when source vector x enormous
 - ✍ Store matrix as sparse $2^k \times 2^l$ blocks
 - ✍ Search over $2^k \times 2^l$ cache blocks to find fastest
-

Register-Blocked Performance of SPMV on Dense Matrices (up to 12x12)

333 MHz Sun Ultra III

Register-Blocked Mat-Vec Performance (Dense 1000x1000) [333 MHz Sun Ultra-III]



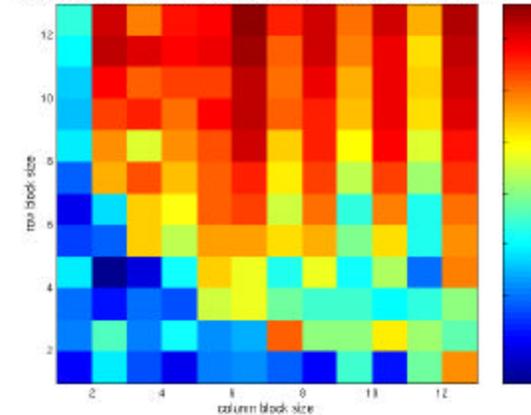
70 Mflops



35 Mflops

800 MHz Pentium III

Register-Blocked Mat-Vec Performance (Dense 1000x1000) [700MHz Intel Pentium III]



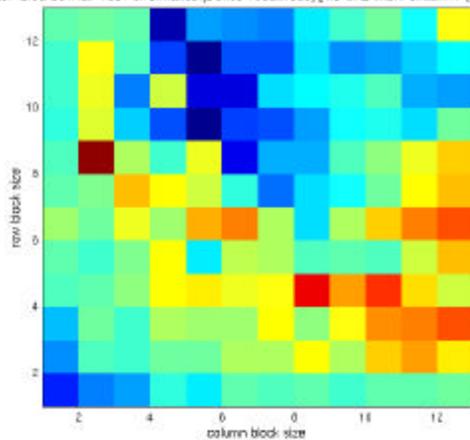
175 Mflops



105 Mflops

1.5 GHz Pentium 4

Register-Blocked Mat-Vec Performance (Dense 1000x1000) [1.5 GHz Intel Pentium IV (Intel C v5.05)]



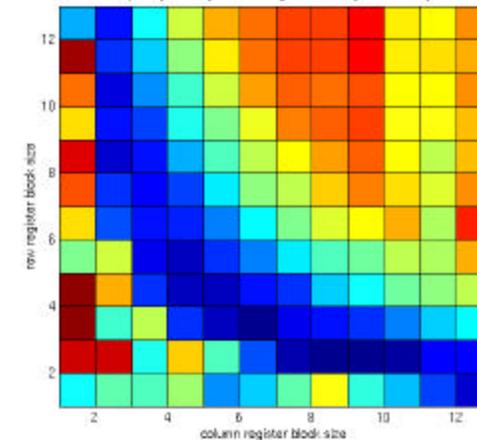
425 Mflops



310 Mflops

800 MHz Itanium

Sparsity Non-symmetric Register Profile (Itanium-icc)



250 Mflops



110 Mflops

Which other sparse operations can we tune?

- ✍ General matrix-vector multiply A^*x
 - ✍ Possibly many vectors x
 - ✍ Symmetric matrix-vector multiply A^*x
 - ✍ Solve a triangular system of equations $T^{-1} * x$
 - ✍ $y = A^T * A * x$
 - ✍ Kernel of Information Retrieval via LSI (SVD)
 - ✍ Same number of memory references as A^*x
 - ✍ $y = ?_i (A(i,:))^T * (A(i,:) * x)$
 - ✍ Future work
 - ✍ $A^2 * x, A^k * x$
 - ✍ Kernel of Information Retrieval used by Google
 - ✍ Includes Jacobi, SOR, ...
 - ✍ Changes calling algorithm
 - ✍ $A^T * M * A$
 - ✍ Matrix triple product
 - ✍ Used in multigrid solver
 - ✍ What does SciDAC need?
-

Test Matrices

✍ General Sparse Matrices

- ✍ Up to $n=76K$, $nnz = 3.21M$
- ✍ From many application areas
- ✍ 1 - Dense
- ✍ 2 to 17 - FEM
- ✍ 18 to 39 - assorted
- ✍ 41 to 44 - linear programming
- ✍ 45 - LSI

✍ Symmetric Matrices

- ✍ Subset of General Matrices
- ✍ 1 - Dense
- ✍ 2 to 8 - FEM
- ✍ 9 to 13 - assorted

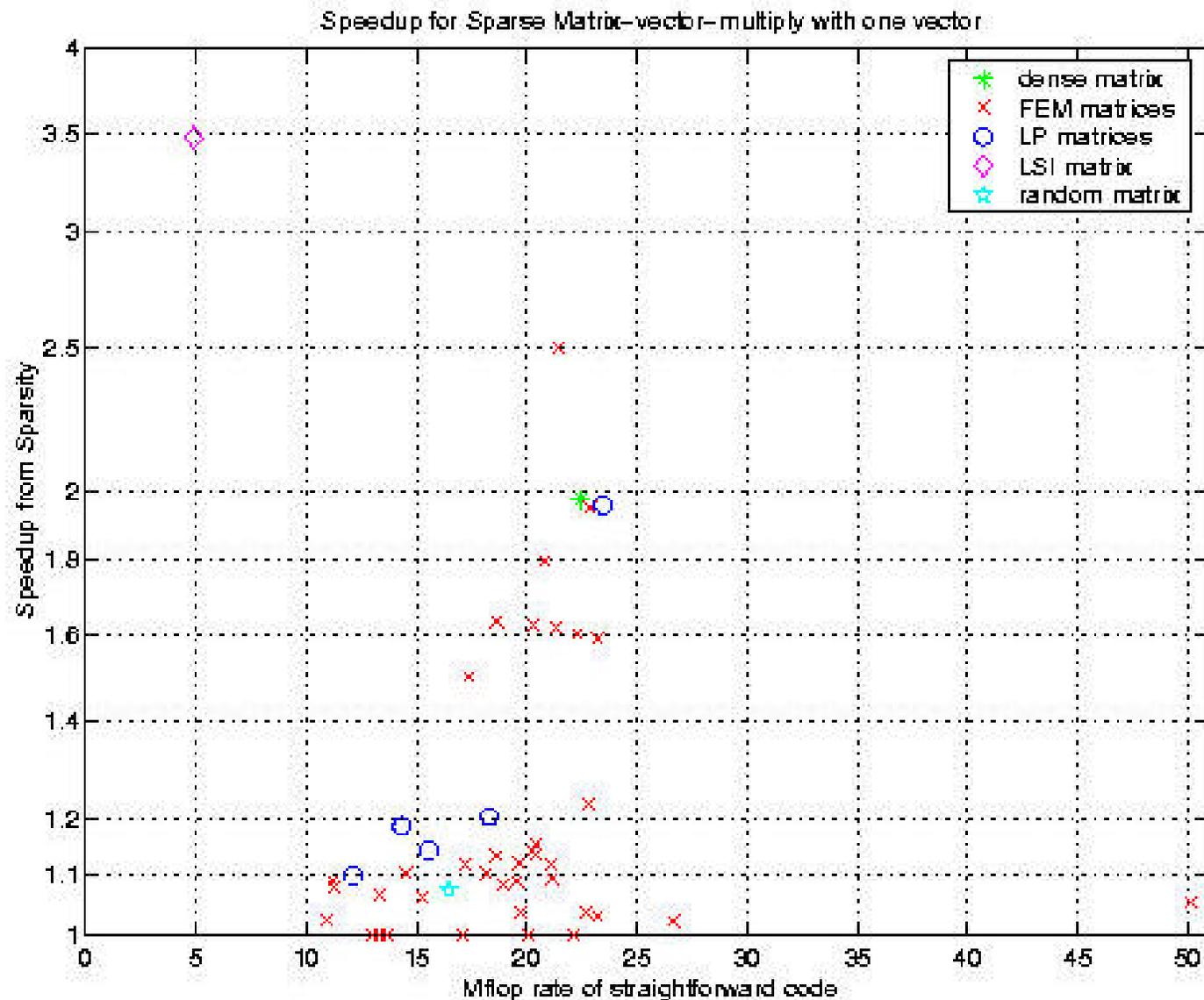
✍ Lower Triangular Matrices

- ✍ Obtained by running SuperLU on subset of General Sparse Matrices
- ✍ 1 - Dense
- ✍ 2 - 13 - FEM

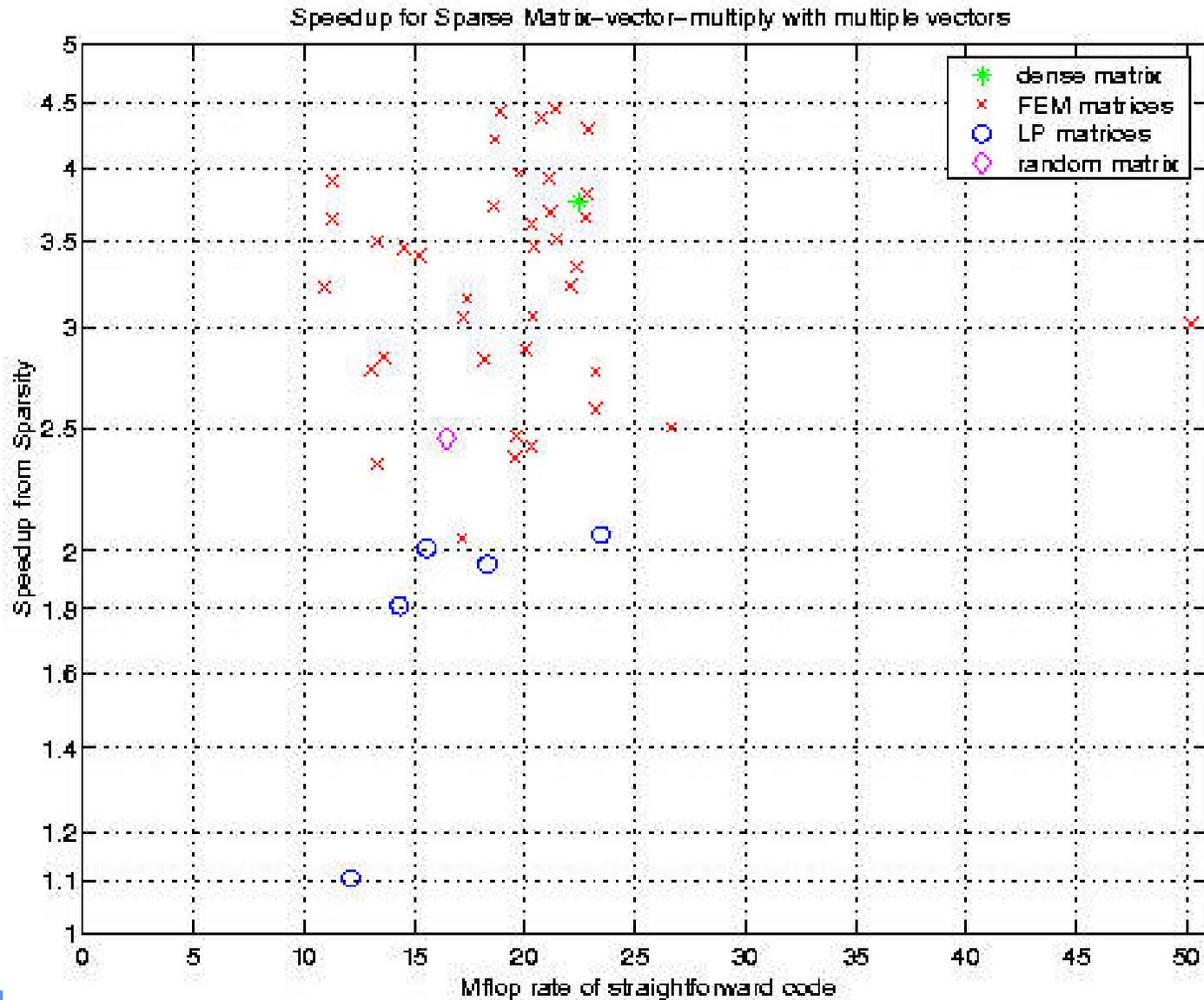
✍ Details on test matrices at end of talk

Results on Sun Ultra 1/170

Speedups on SPMV from Sparsity on Sun Ultra 1/170 - 1 RHS

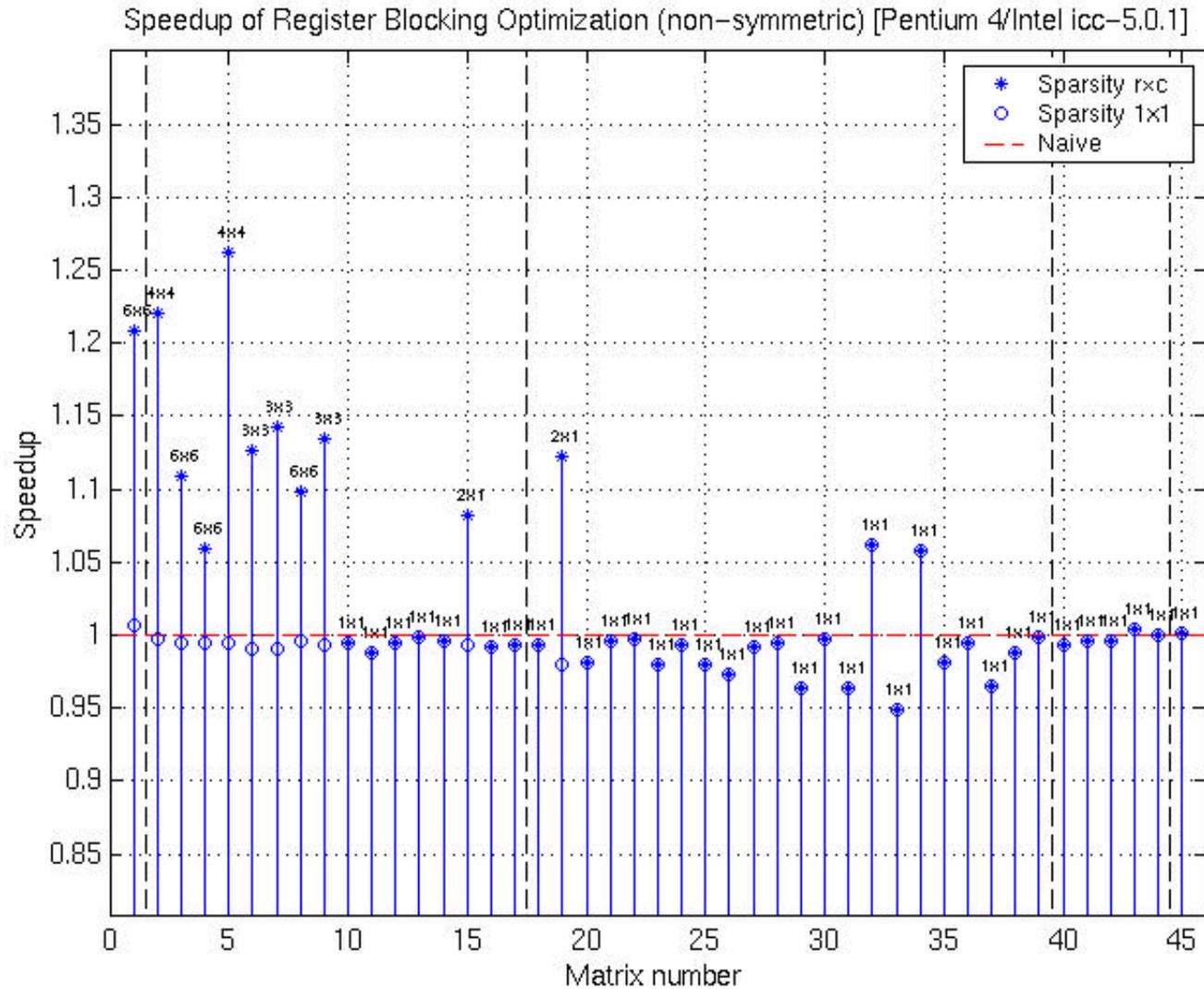


Speedups on SPMV from Sparsity on Sun Ultra 1/170 - 9 RHS

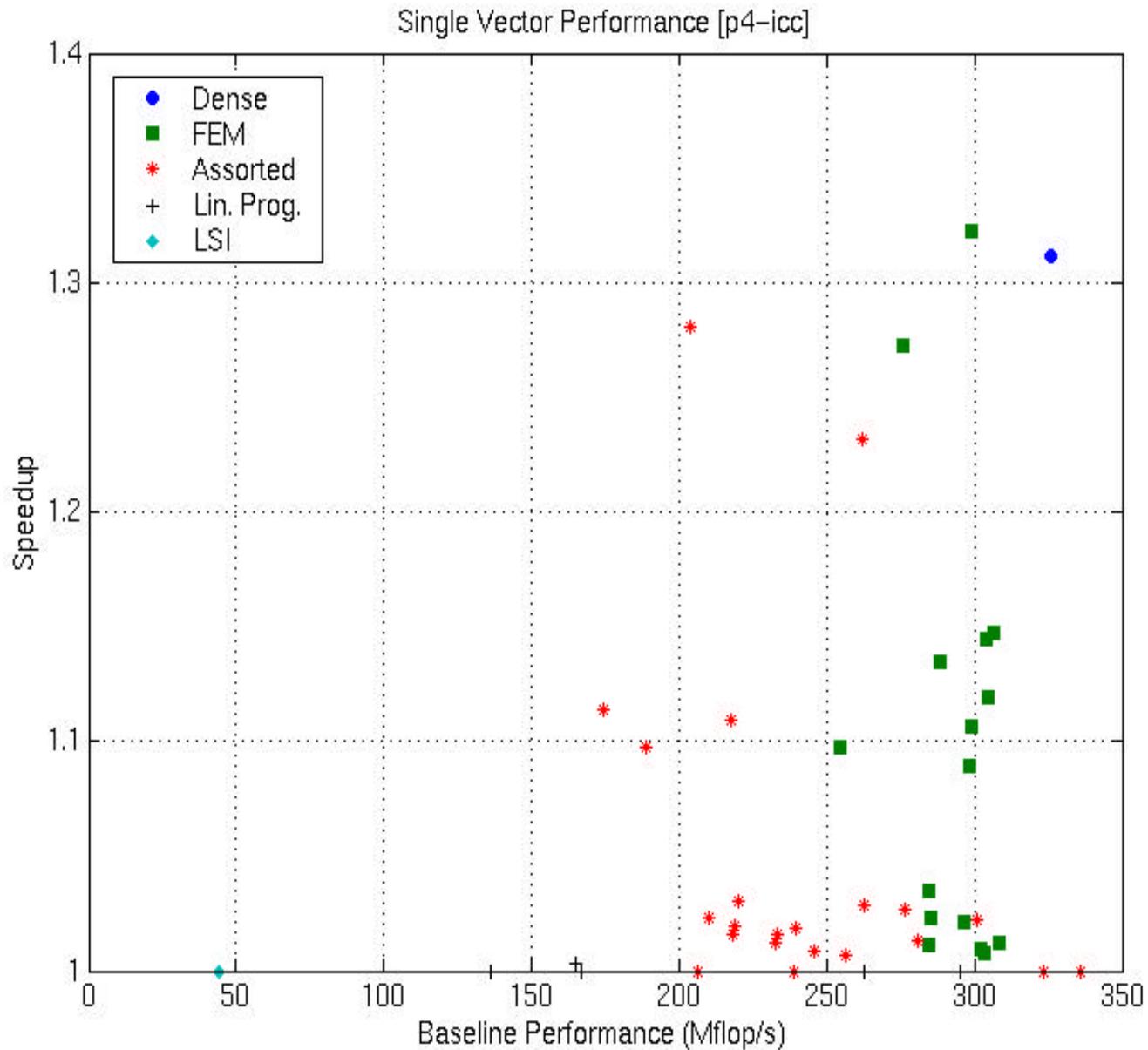


***Recent Results
on P4 using icc and gcc***

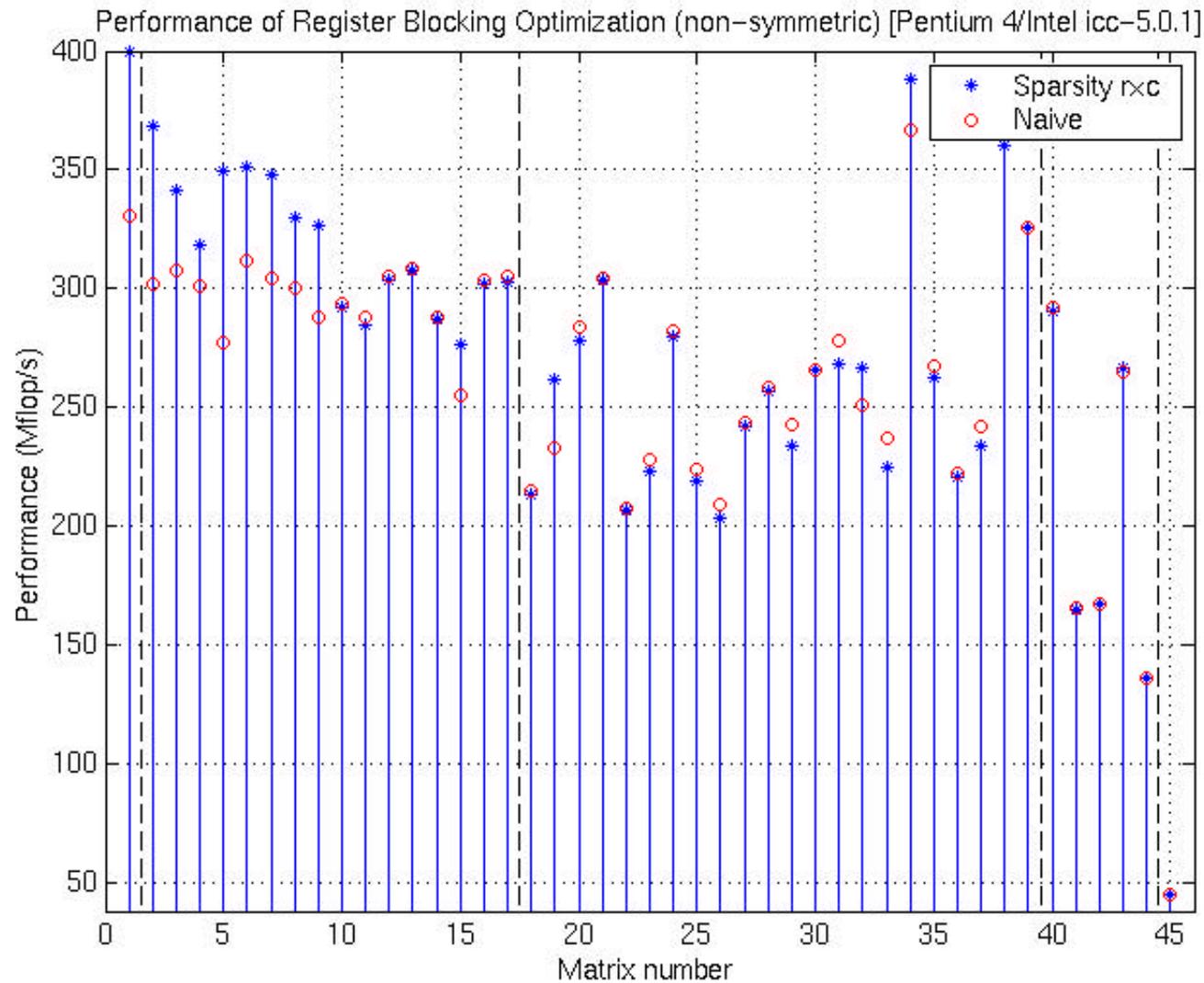
Speedup of SPMV from Sparsity on P4/icc-5.0.1



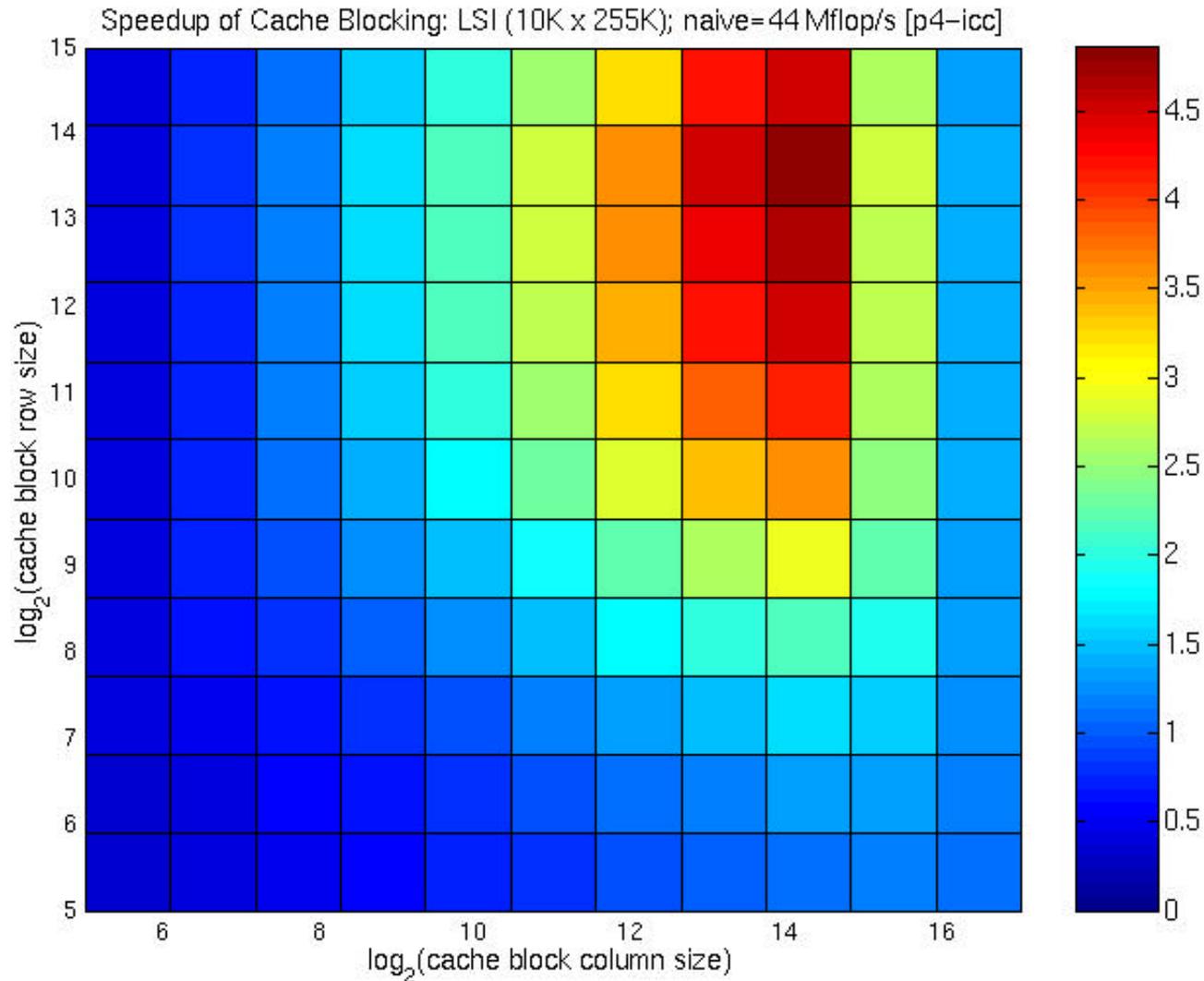
Single vector speedups on P4 by matrix type - best r x c



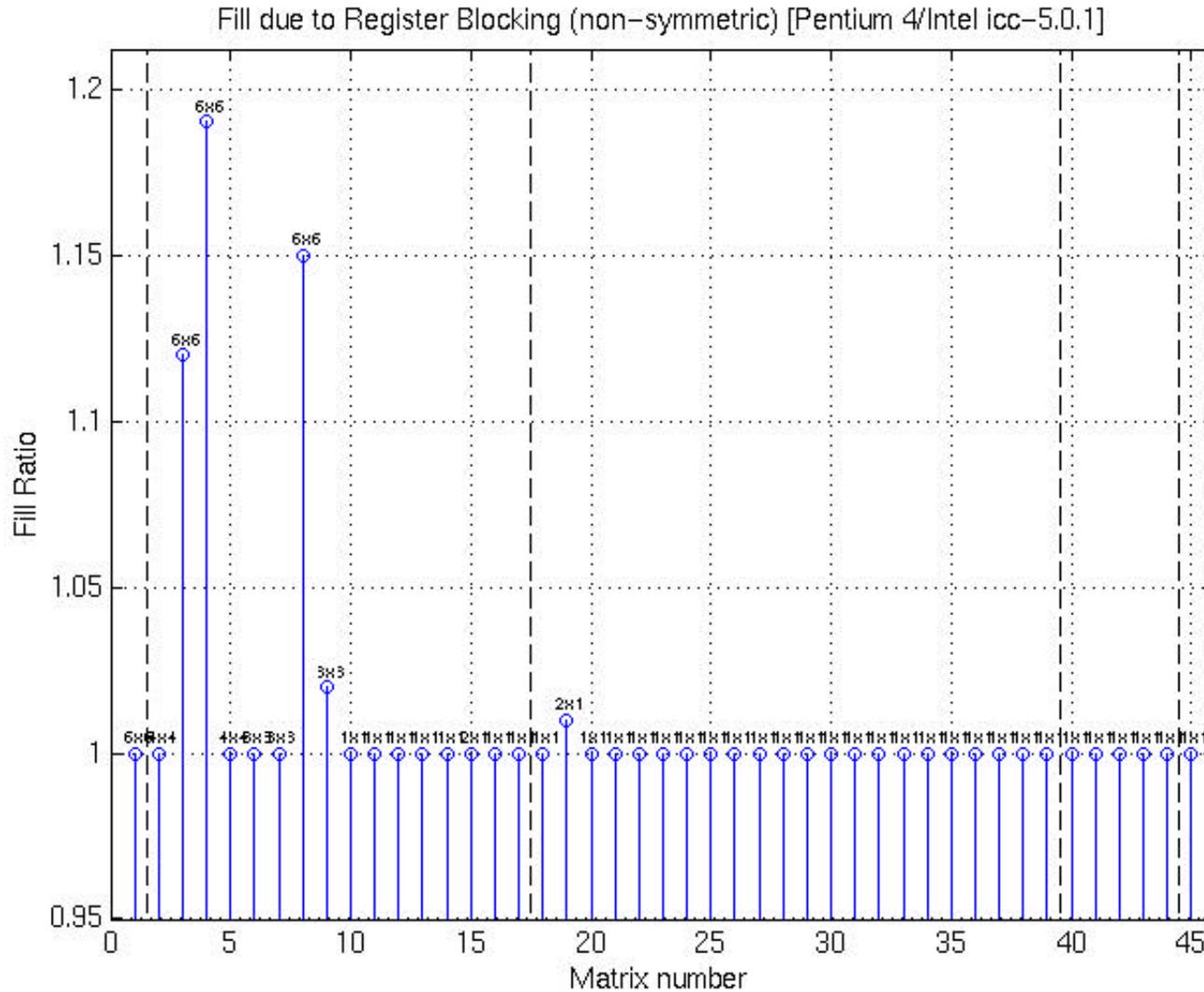
Performance of SPMV from Sparsity on P4/icc-5.0.1



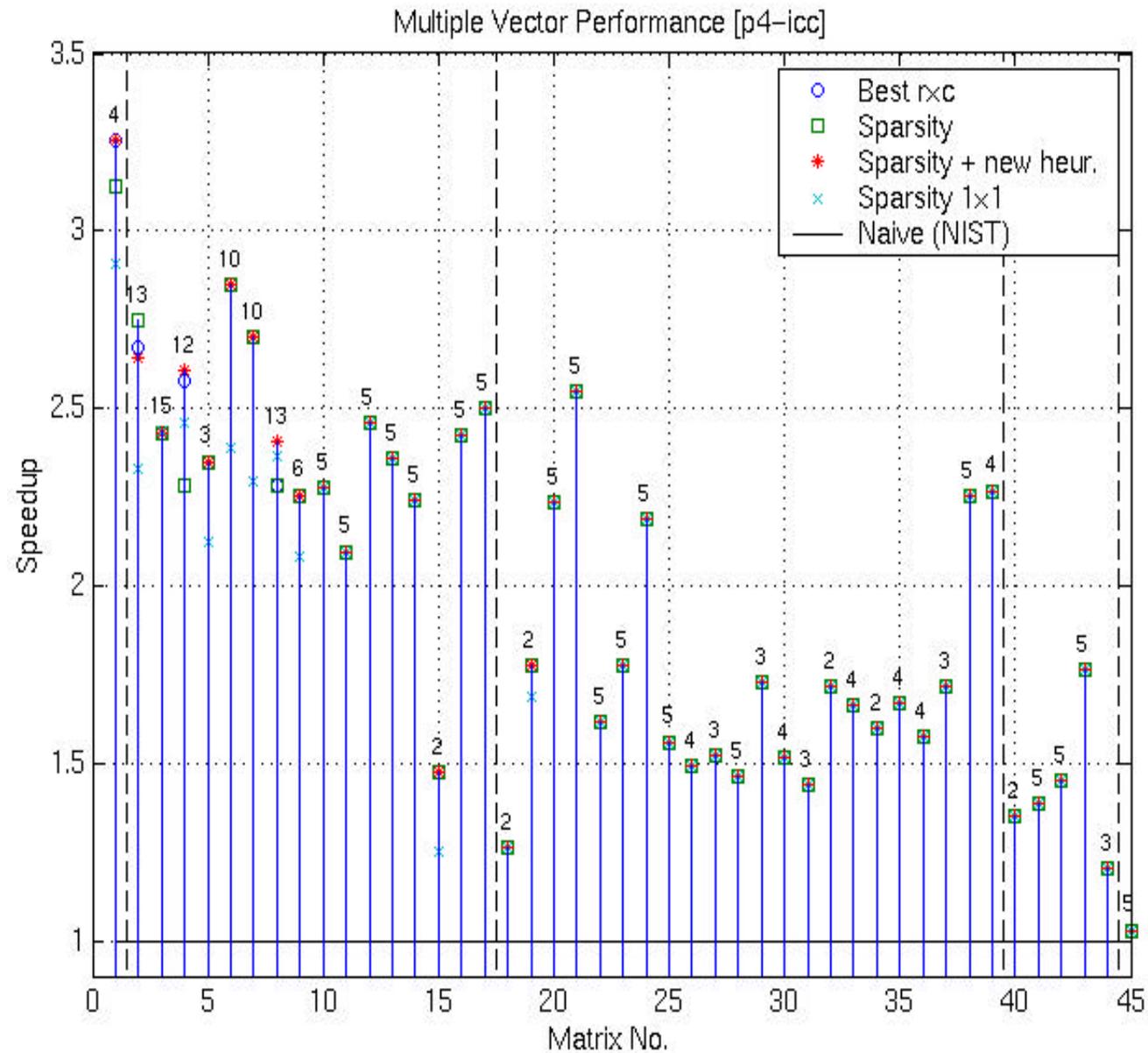
Speed up from Cache Blocking on LSI matrix on P4



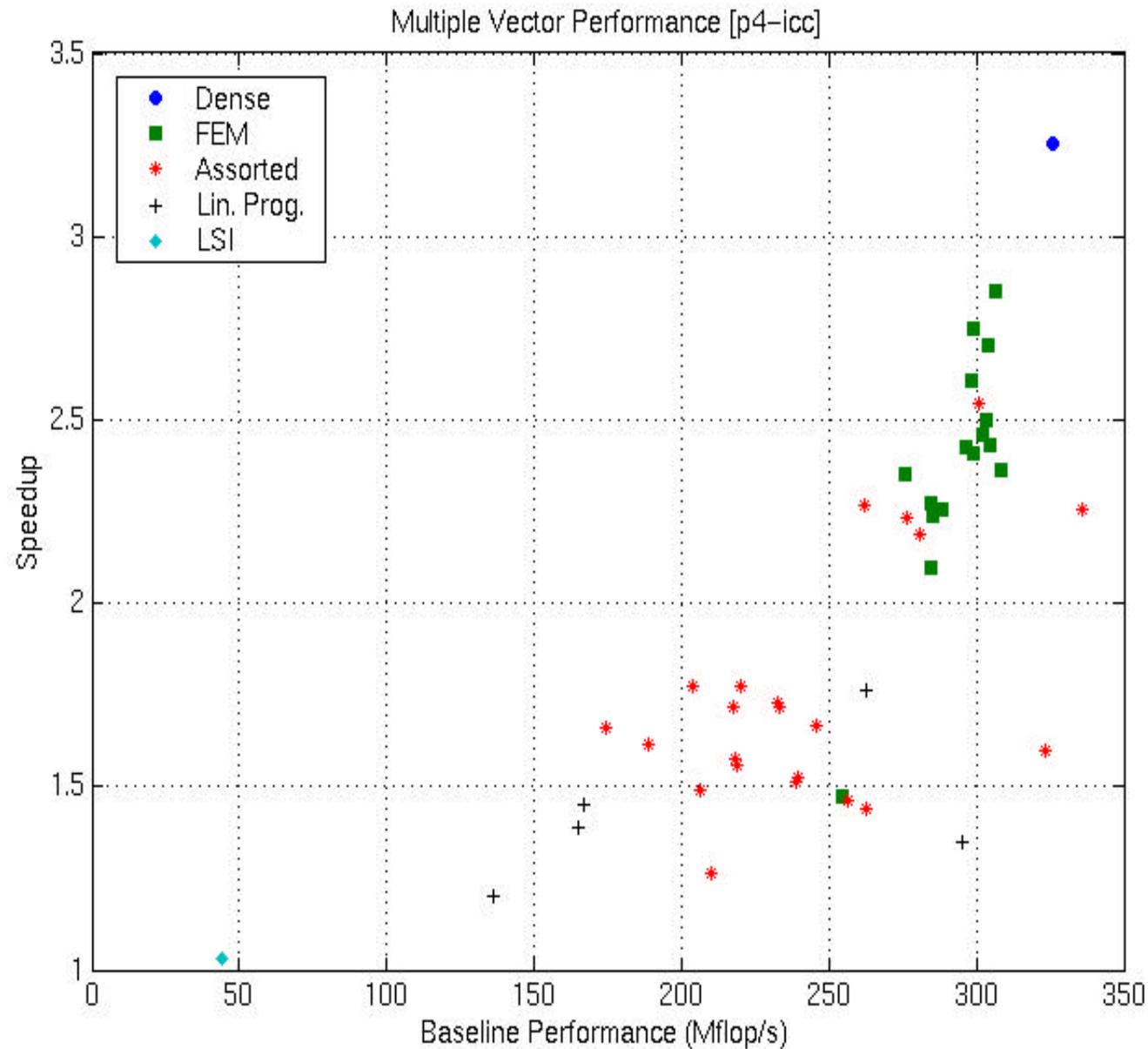
Fill for SPMV from Sparsity on P4/icc-5.0.1



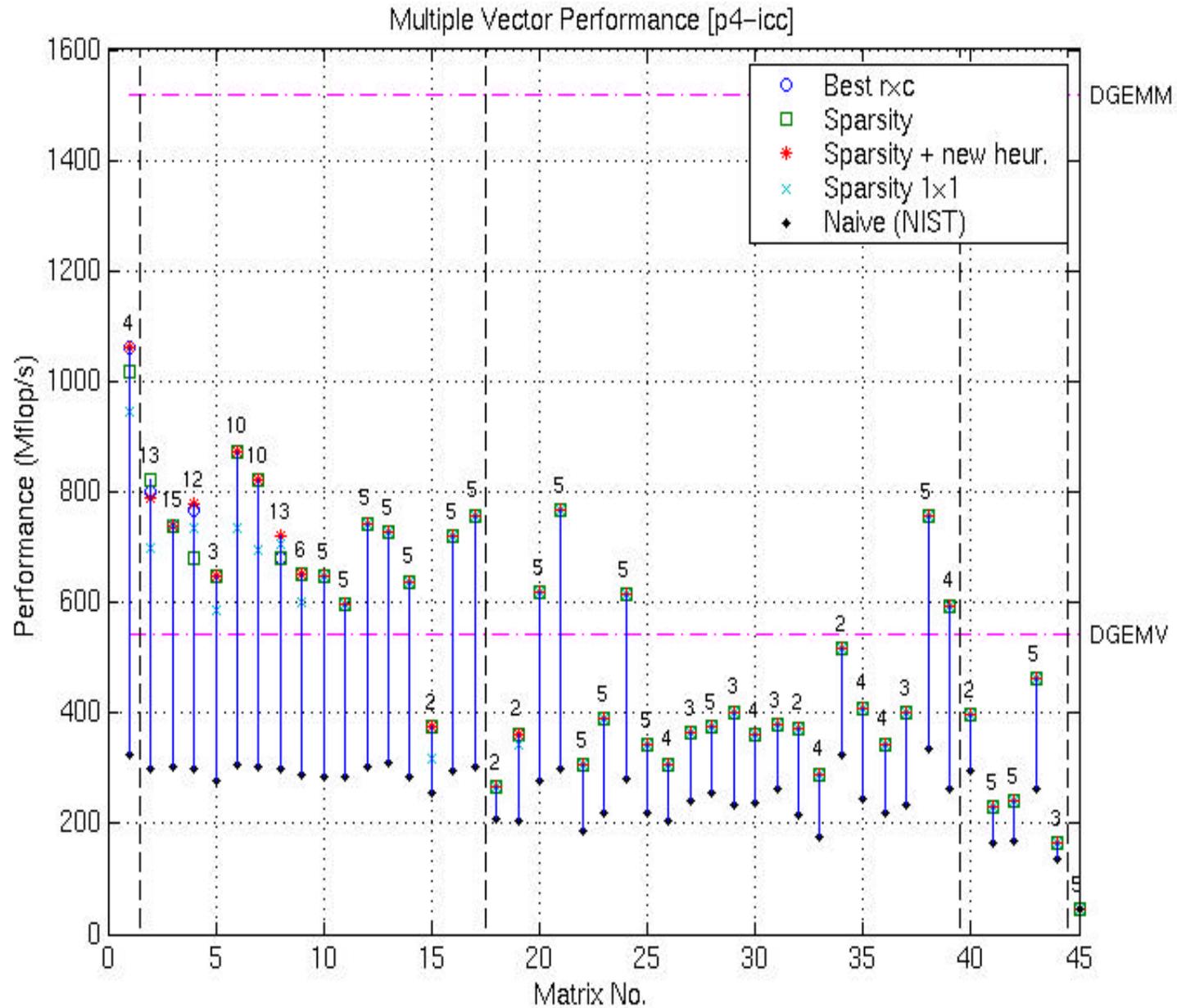
Multiple vector speedups on P4



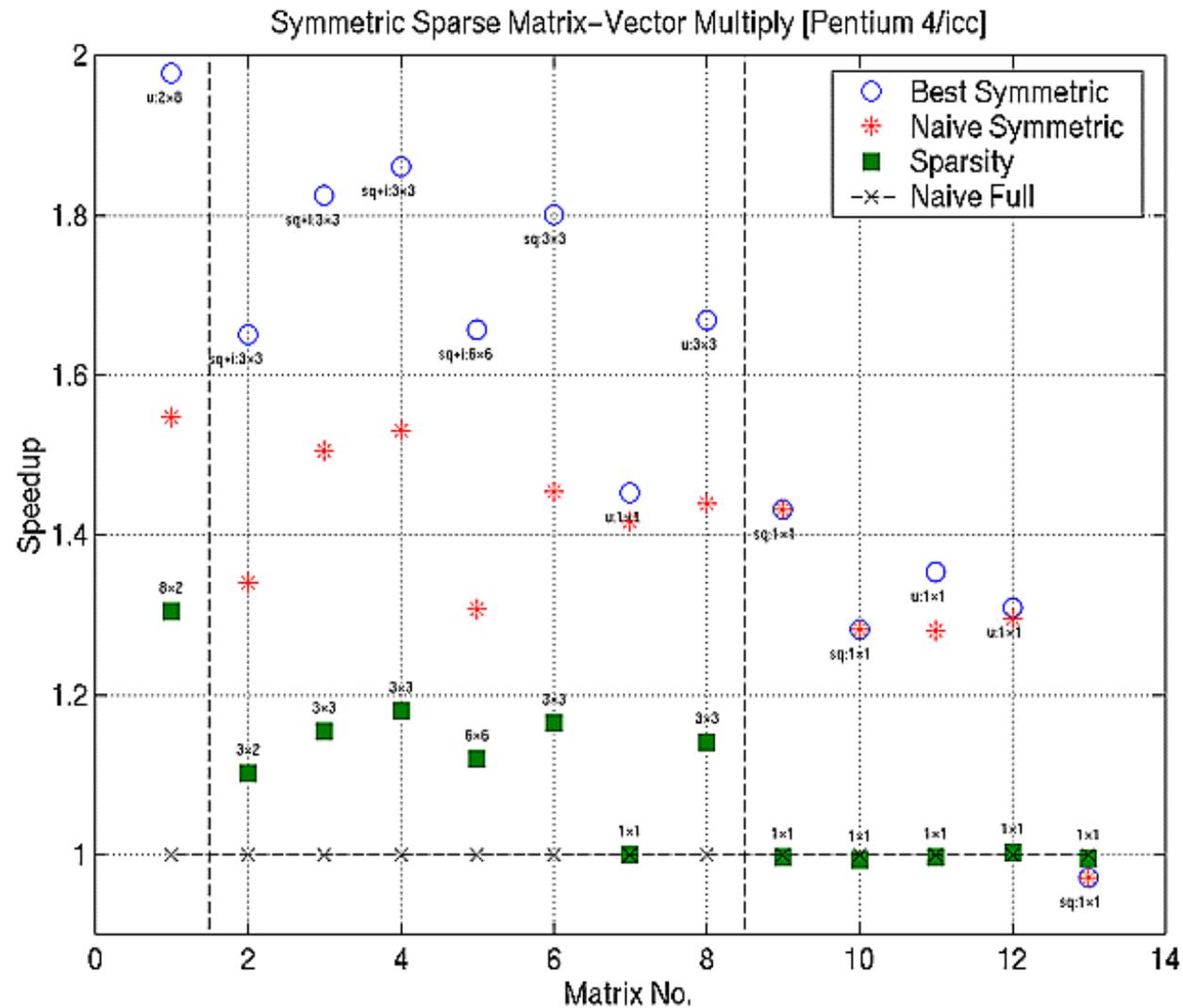
Multiple vector speedups on P4 - by matrix type



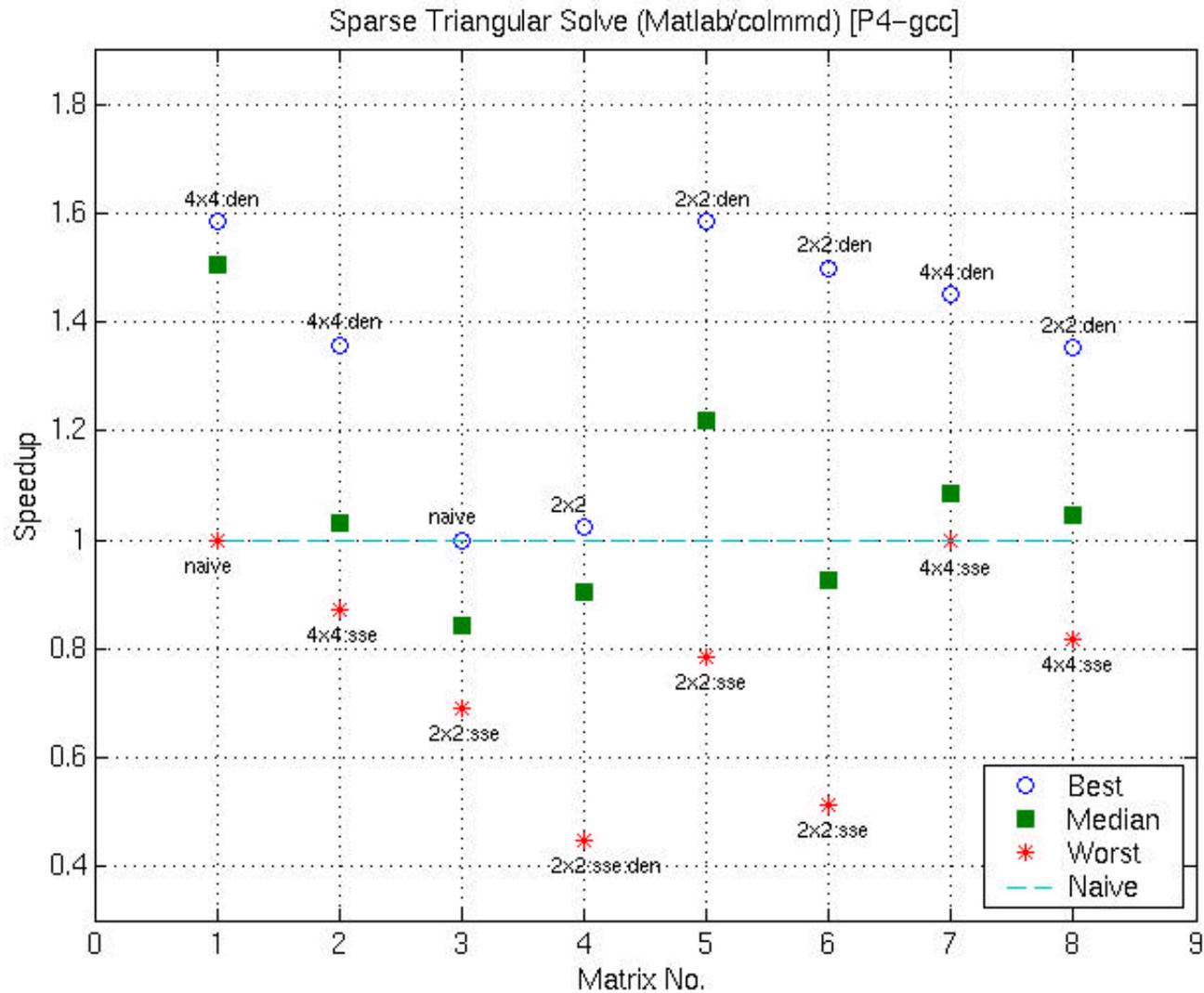
Multiple Vector Performance on P4



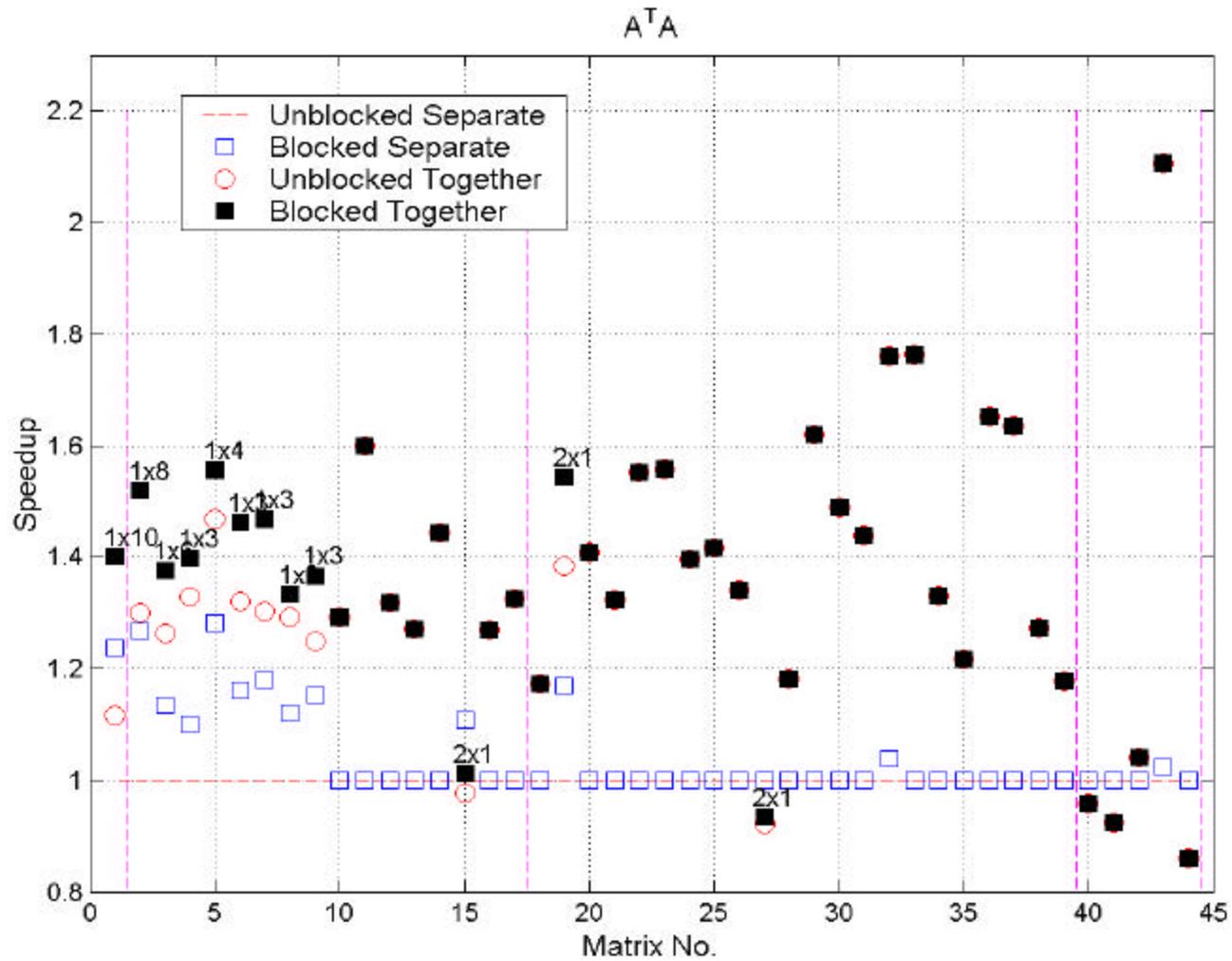
Symmetric Sparse Matrix-Vector Multiply on P4 (vs naive full = 1)



Sparse Triangular Solve (Matlab's colmmd ordering) on P4

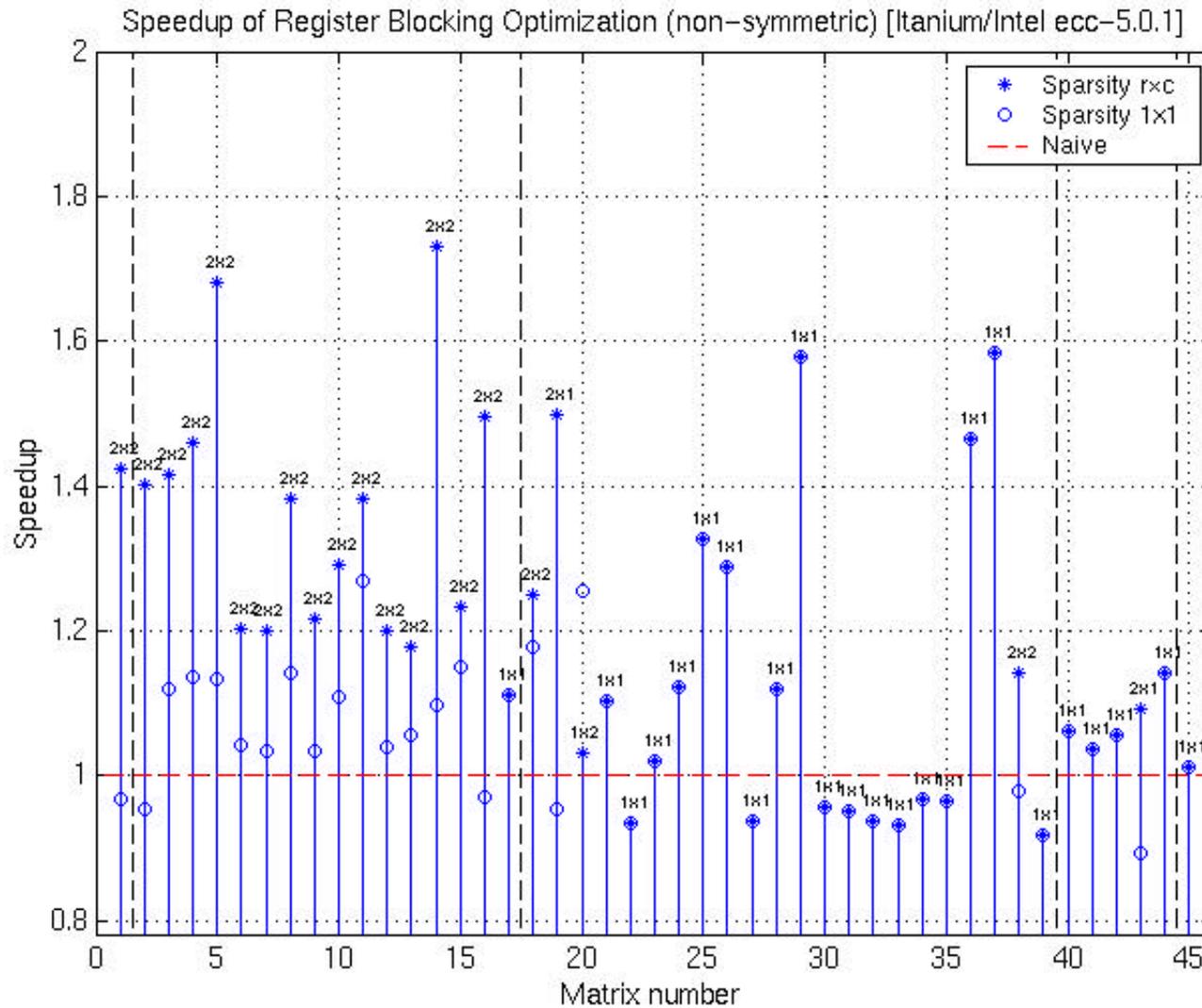


$A^T * A$ on P4 (Accesses A only once)

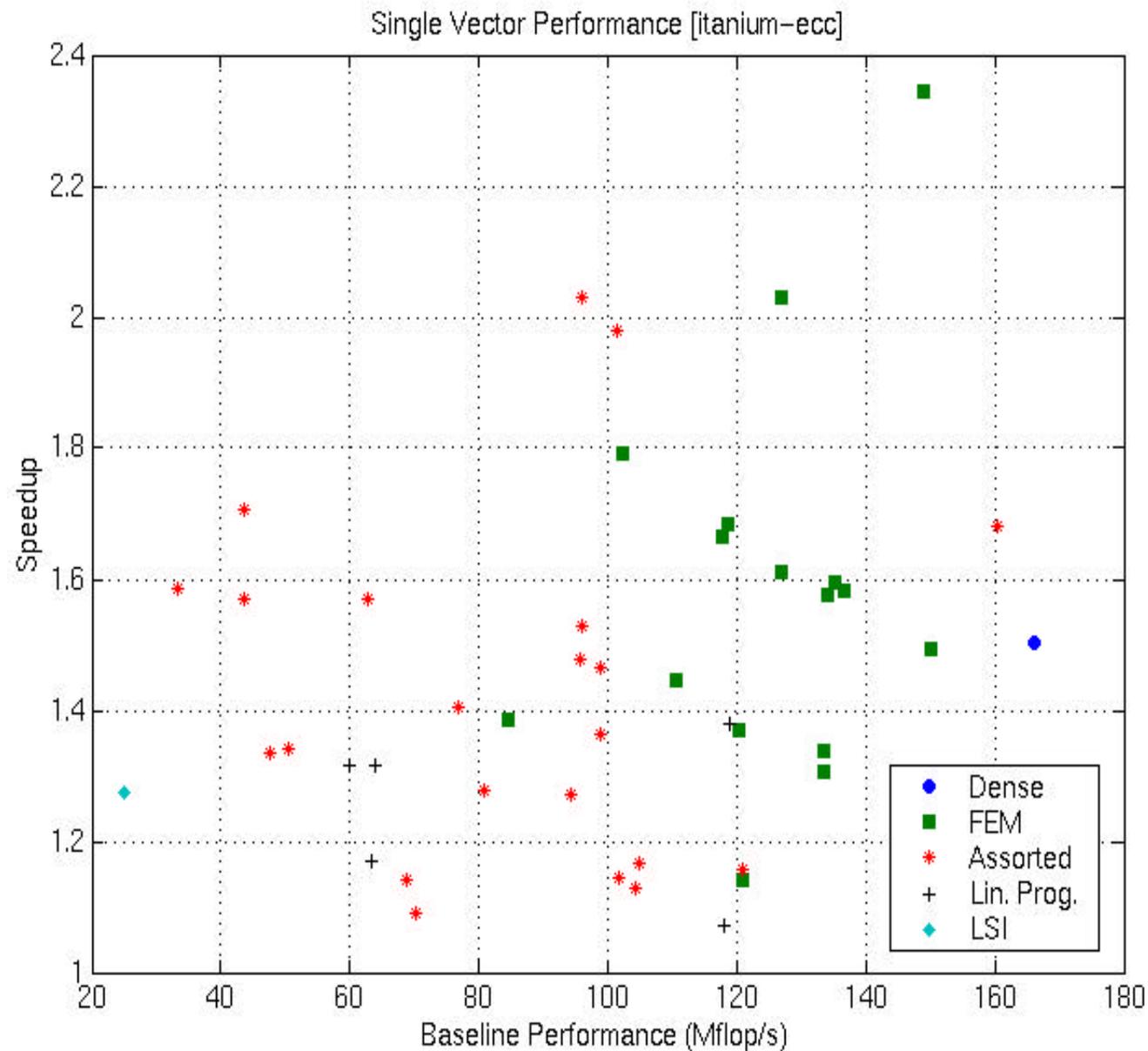


***Preliminary Results on
Titanium using ecc***

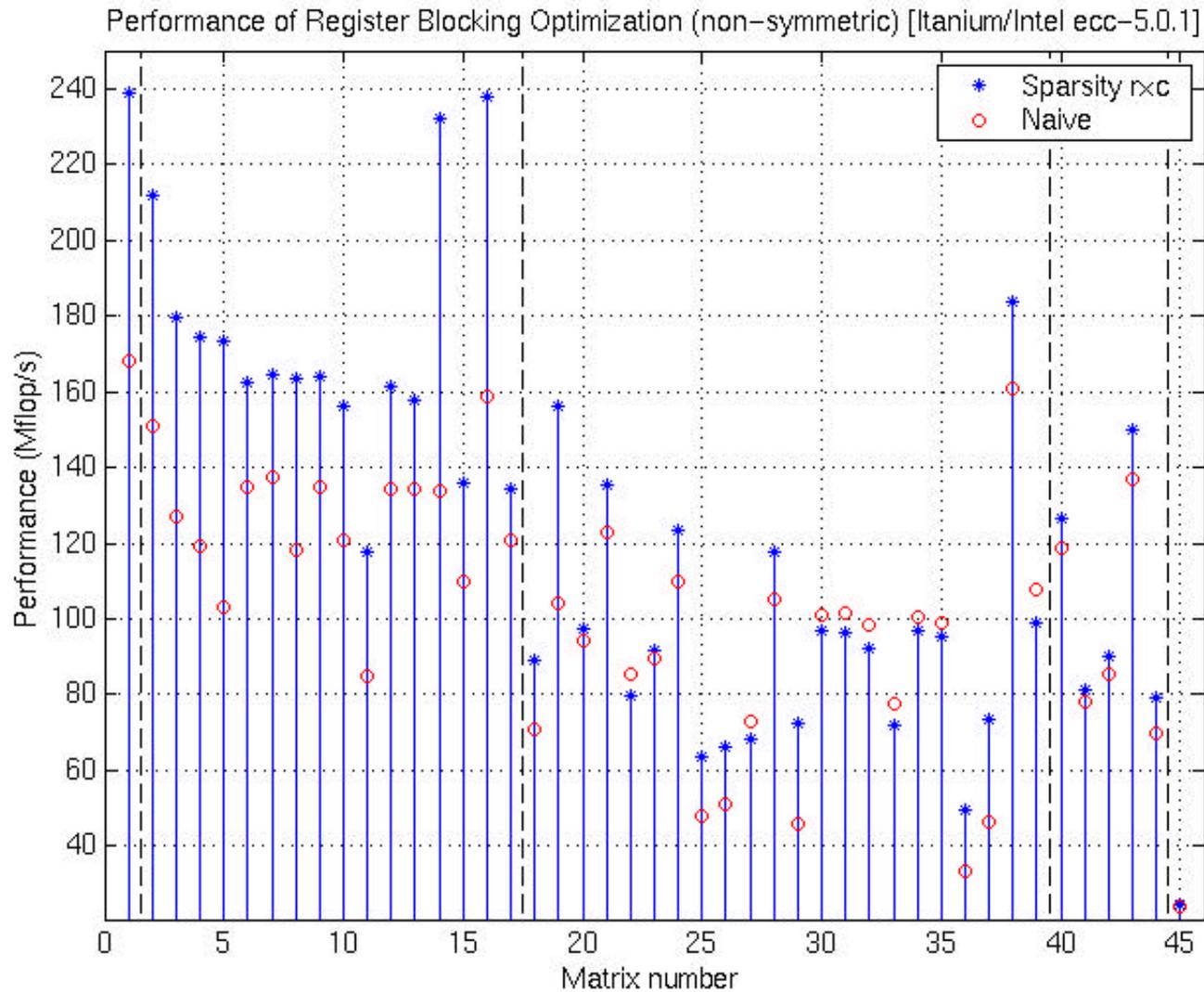
Speedup of SPMV from Sparsity on Itanium/ecc-5.0.1



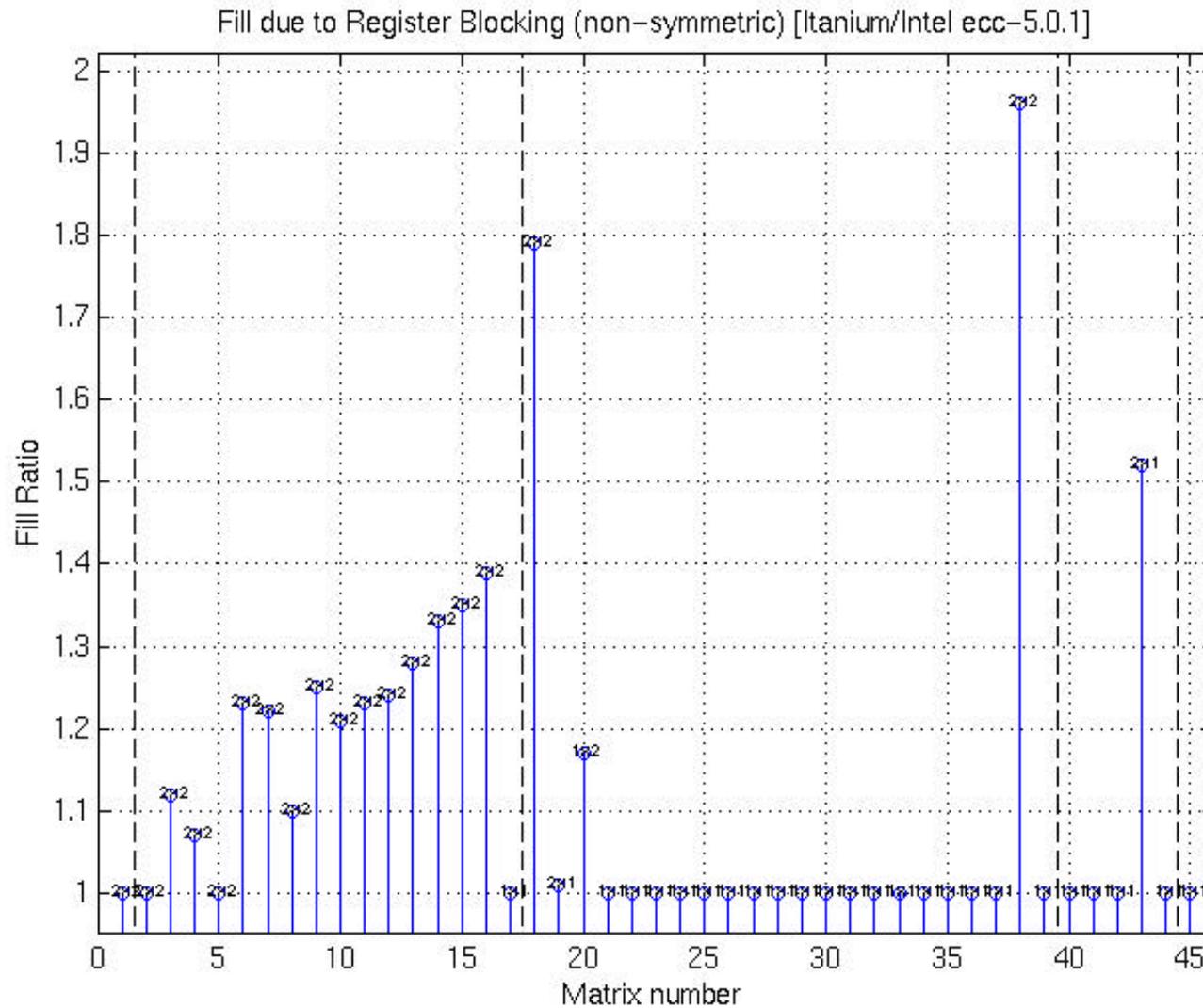
Single vector speedups on Itanium by matrix type



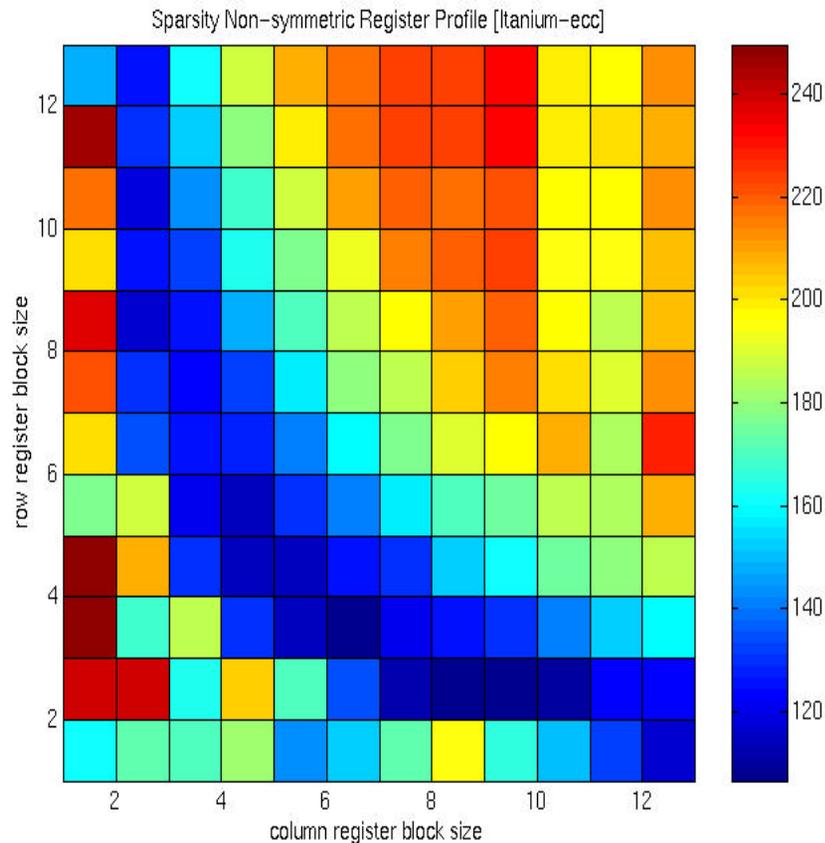
Raw Performance of SPMV from Sparsity on Itanium



Fill for SPMV from Sparsity on Itanium



Improvements to register block size selection



✍ Initial heuristic to determine best $r \times c$ block biased to diagonal of performance plot

✍ Didn't matter on Sun, does on P4 and Itanium since performance so "nondiagonally dominant"

✍ Matrix 8:

✍ Chose 2x2 (164 Mflops)

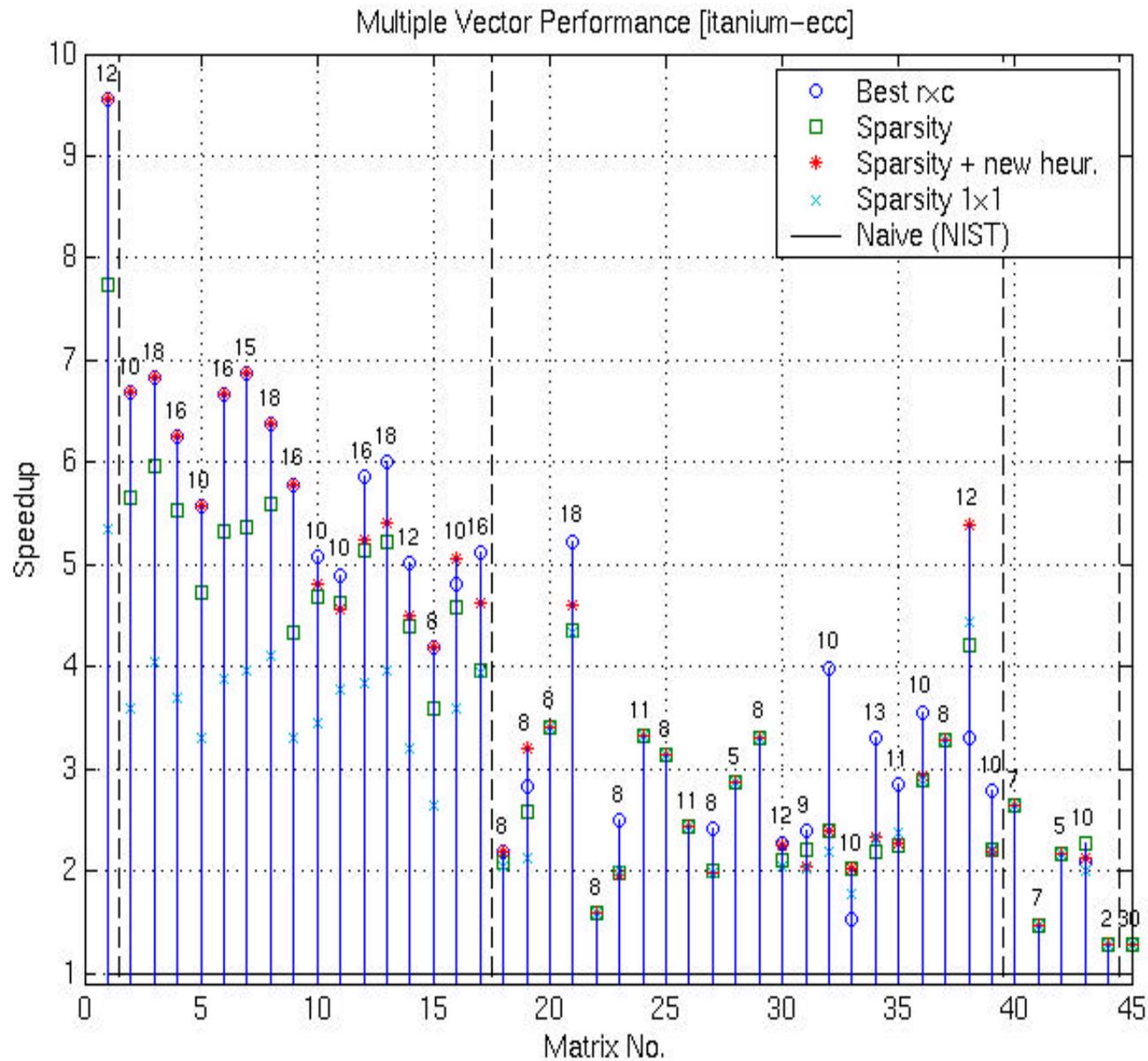
✍ Better: 3x1 (196 Mflops)

✍ Matrix 9:

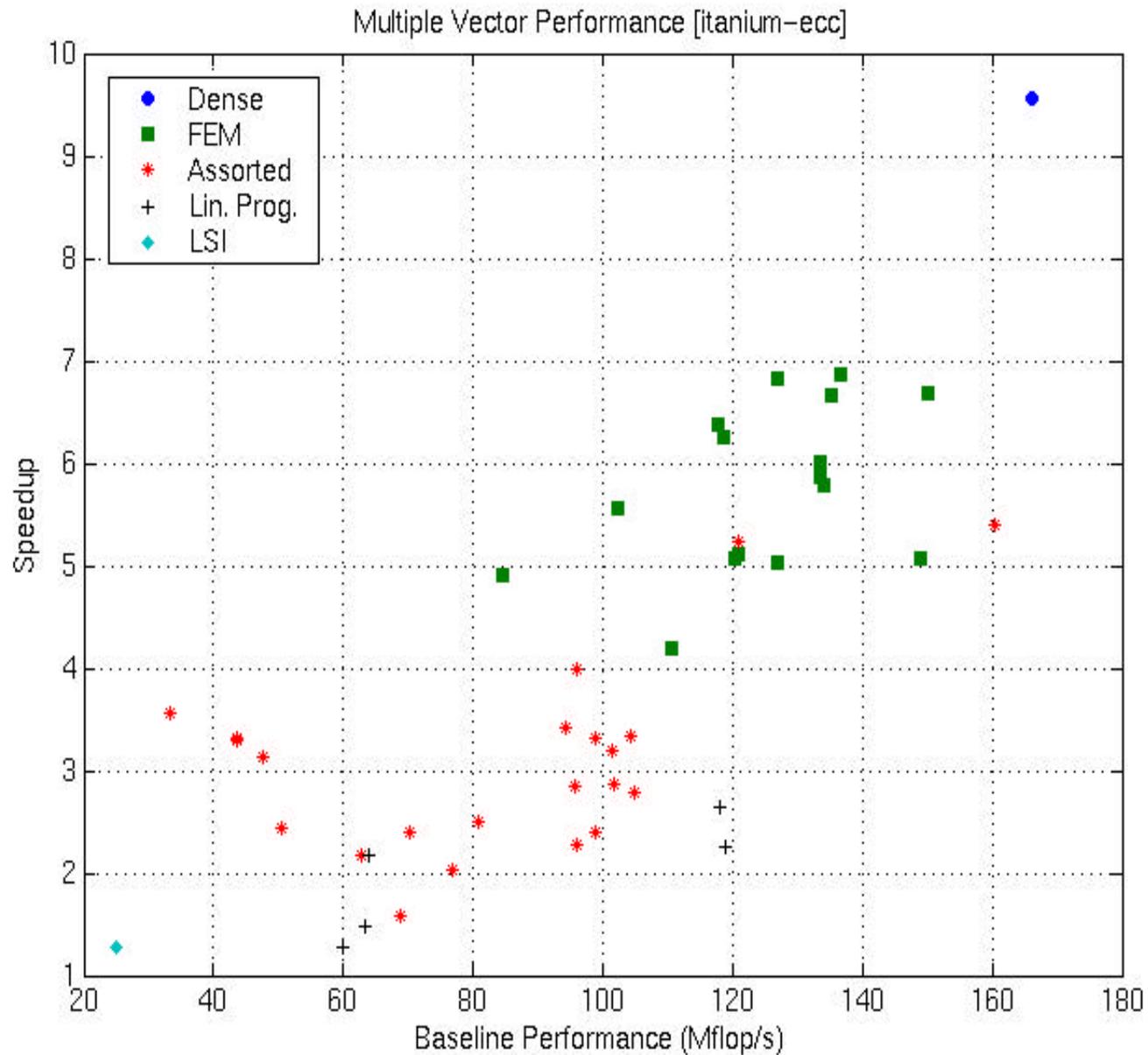
✍ Chose 2x2 (164 Mflops)

✍ Better: 3x1 (213 Mflops)

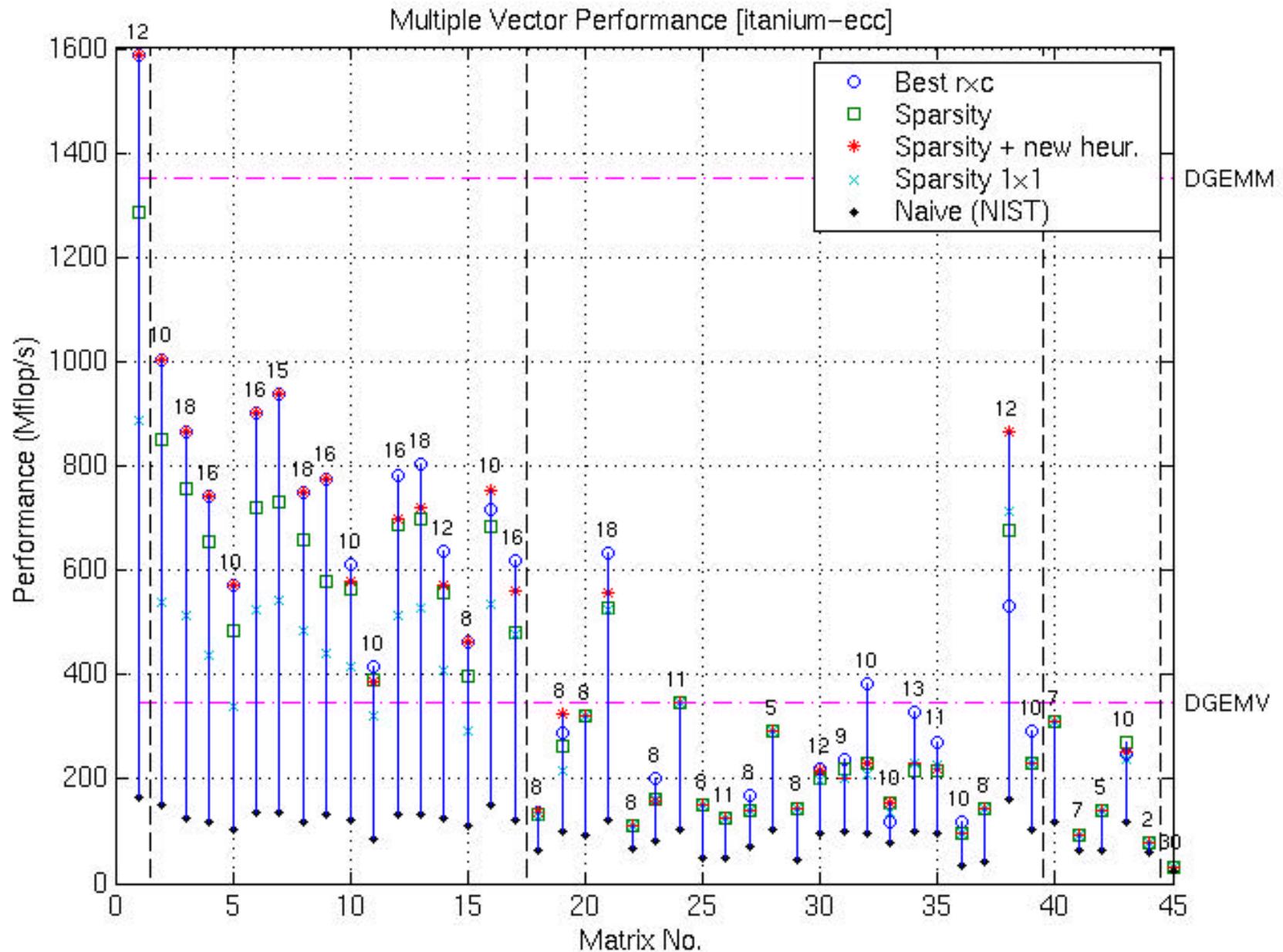
Multiple vector speedups on Itanium



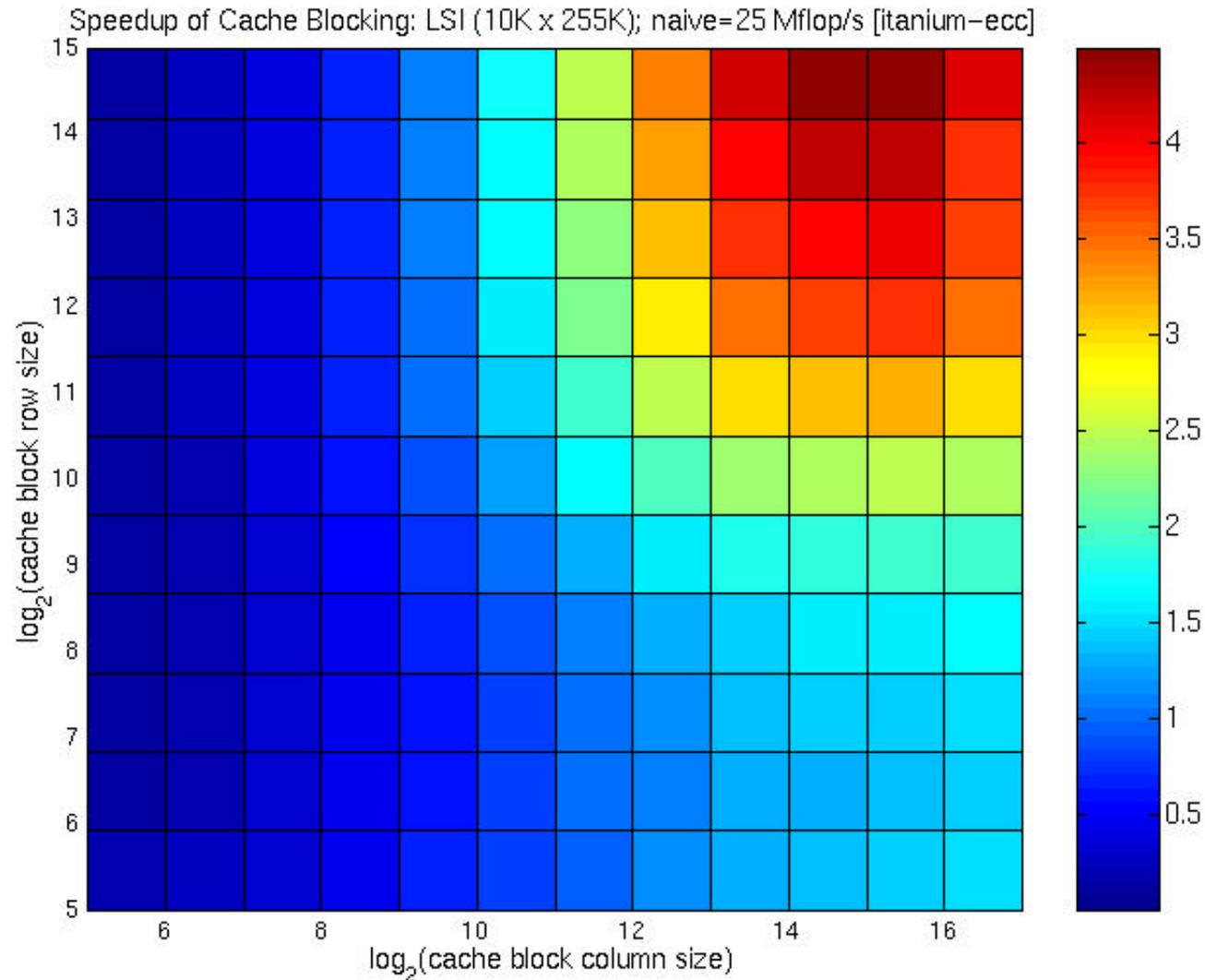
Multiple vector speedups on Itanium - by matrix type



Multiple Vector Performance on Itanium



Speed up from Cache Blocking on LSI matrix on Itanium



Future Work

Exploit New Architectures

Itanium

-  128 (82-bit) floating point registers
-  fused multiply-add instruction
-  predicated instructions
-  rotating registers for software pipelining
-  prefetch instructions
-  three levels of cache

McKinley

Tune current and wider set of kernels

-  Improve heuristics, eg choice of $r \times c$

Incorporate into applications

-  PETSC, library, ...

Further automate performance tuning

-  Generation of algorithm space generators
-

Background on Test Matrices

Sparse Matrix Benchmark Suite (1/3)

#	Matrix Name	Problem Domain	Dimension	No. Non-zeros
1	dense	Dense matrix	1,000	1.00 M
2	raefsky3	Fluid structure interaction	21,200	1.49 M
3	inaccura	Accuracy problem	16,146	1.02 M
4	bcsstk35*	Stiff matrix automobile frame	30,237	1.45 M
5	venkat01	Flow simulation	62,424	1.72 M
6	crystk02*	FEM crystal free-vibration	13,965	969 k
7	crystk03*	FEM crystal free-vibration	24,696	1.75 M
8	nasasrb*	Shuttle rocket booster	54,870	2.68 M
9	3dtube*	3-D pressure tube	45,330	3.21 M
10	ct20stif*	CT20 engine block	52,329	2.70 M
11	bai	Airfoil eigenvalue calculation	23,560	484 k
12	raefsky4	Buckling problem	19,779	1.33 M
13	ex11	3-D steady flow problem	16,214	1.10 M
14	rdist1	Chemical process simulation	4,134	94.4 k
15	vavasis3	2-D PDE problem	41,092	1.68 M

Note: * indicates a symmetric matrix.

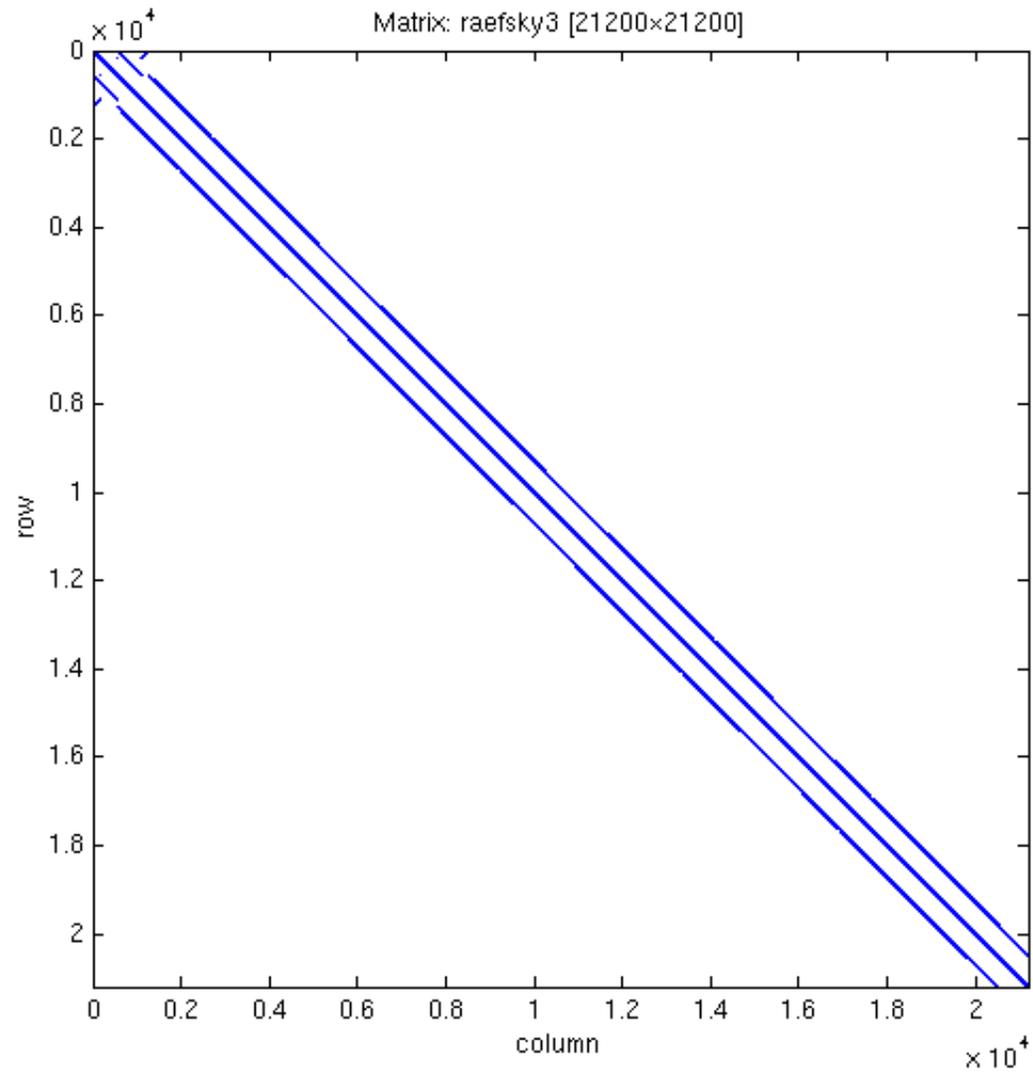
Sparse Matrix Benchmark Suite (2/3)

#	Matrix Name	Problem Domain	Dimension	No. Non-zeros
16	orani678	Economic modeling	2,529	90.2 k
17	rim	FEM fluid mechanics problem	22,560	1.01 M
18	memplus	Circuit simulation	17,758	126 k
19	gemat11	Power flow	4,929	33.1 k
20	lhr10	Chemical process simulation	10,672	233 k
21	goodwin*	Fluid mechanics problem	7,320	325 k
22	bayer02	Chemical process simulation	13,935	63.7 k
23	bayer10	Chemical process simulation	13,436	94.9 k
24	coater2	Simulation of coating flows	9,540	207 k
25	finan512*	Financial portfolio optimization	74,752	597 k
26	onetone2	Harmonic balance method	36,057	228 k
27	pwt*	Structural engineering	36,519	326 k
28	vibrobox*	Vibroacoustics	12,328	343 k
29	wang4	Semiconductor device simulation	26,068	177 k
30	Insp3937	Fluid flow modeling	3,937	25.4 k

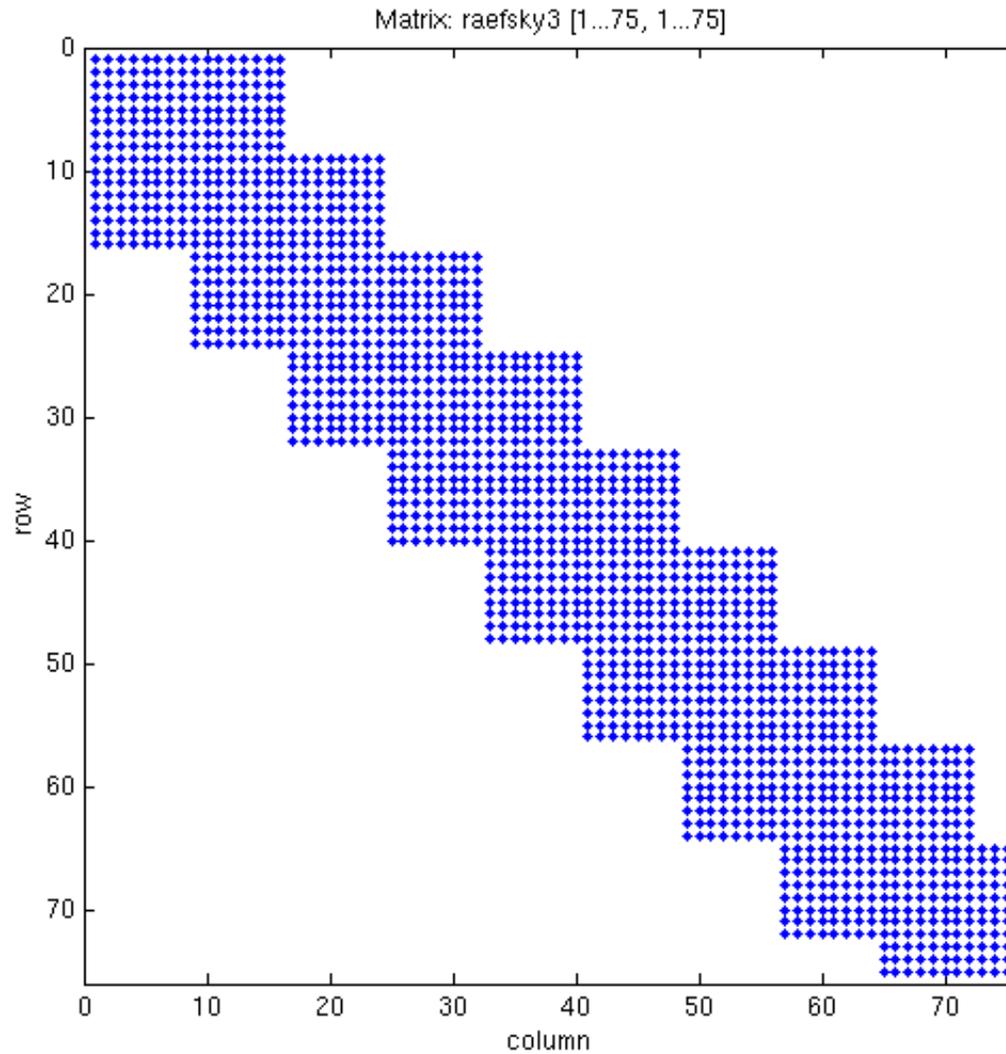
Sparse Matrix Benchmark Suite (3/3)

#	Matrix Name	Problem Domain	Dimensions	No. Non-zeros
31	Ins3937	Fluid flow modeling	3,937	25.4 k
32	sherman5	Oil reservoir modeling	3,312	20.8 k
33	sherman3	Oil reservoir modeling	5,005	20.0 k
34	orsreg1	Oil reservoir modeling	2,205	14.1 k
35	saylr4	Oil reservoir modeling	3,564	22.3 k
36	shyy161	Viscous flow calculation	76,480	330 k
37	wang3	Semiconductor device simulation	26,064	177 k
38	mcfe	Astrophysics	765	24.4 k
39	jpwh991	Circuit physics problem	991	6,027
40	gupta1*	Linear programming	31,802	2.16 M
41	lpcreb	Linear programming	9,648 x 77,137	261 k
42	lpcred	Linear programming	8,926 x 73,948	247 k
43	lpfit2p	Linear programming	3,000 x 13,525	50.3 k
44	lpnug20	Linear programming	15,240 x 72,600	305 k
45	lsi	Latent semantic indexing	10 k x 255 k	3.7 M

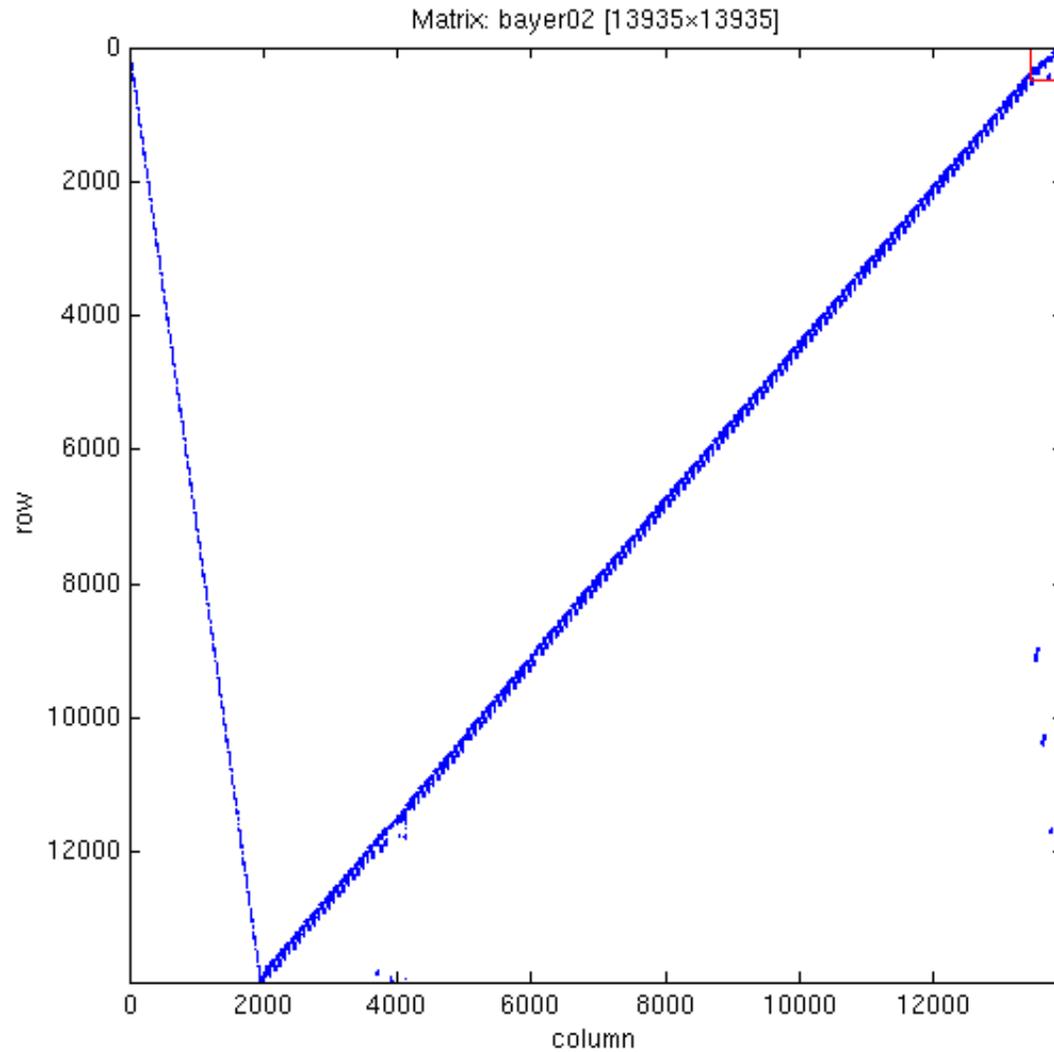
Matrix #2 - raefsky3 (FEM/Fluids)



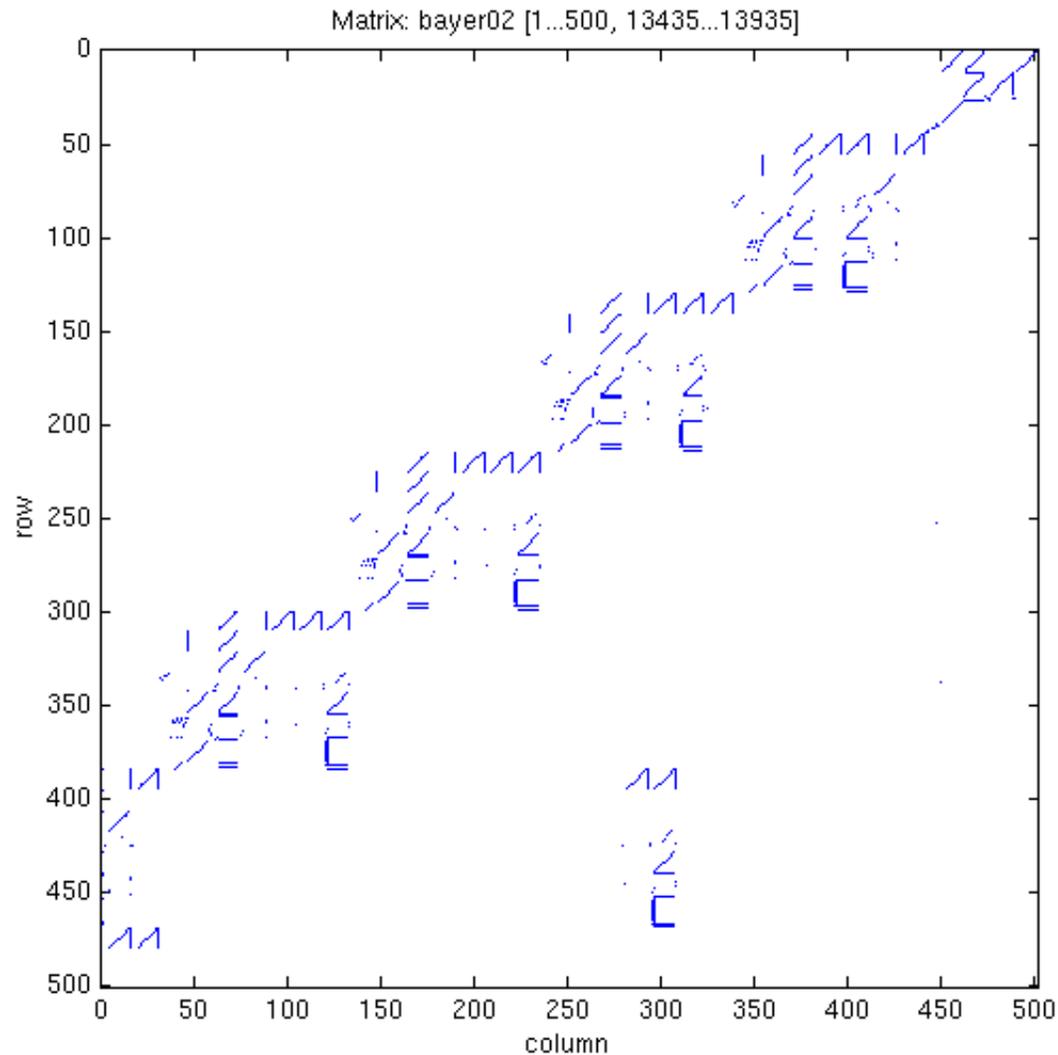
Matrix #2 (cont'd) - raefsky3 (FEM/Fluids)



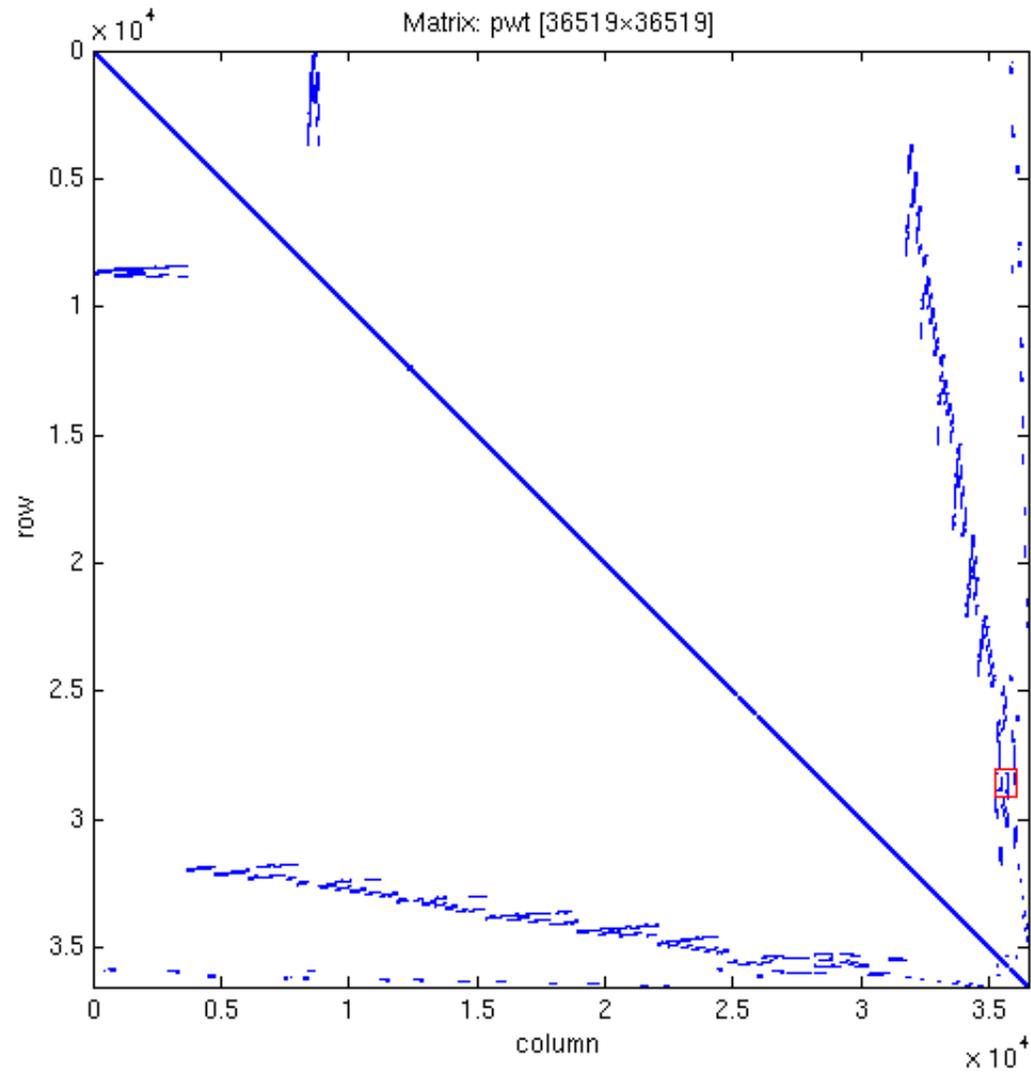
Matrix #22 - bayer02 (chemical process simulation)



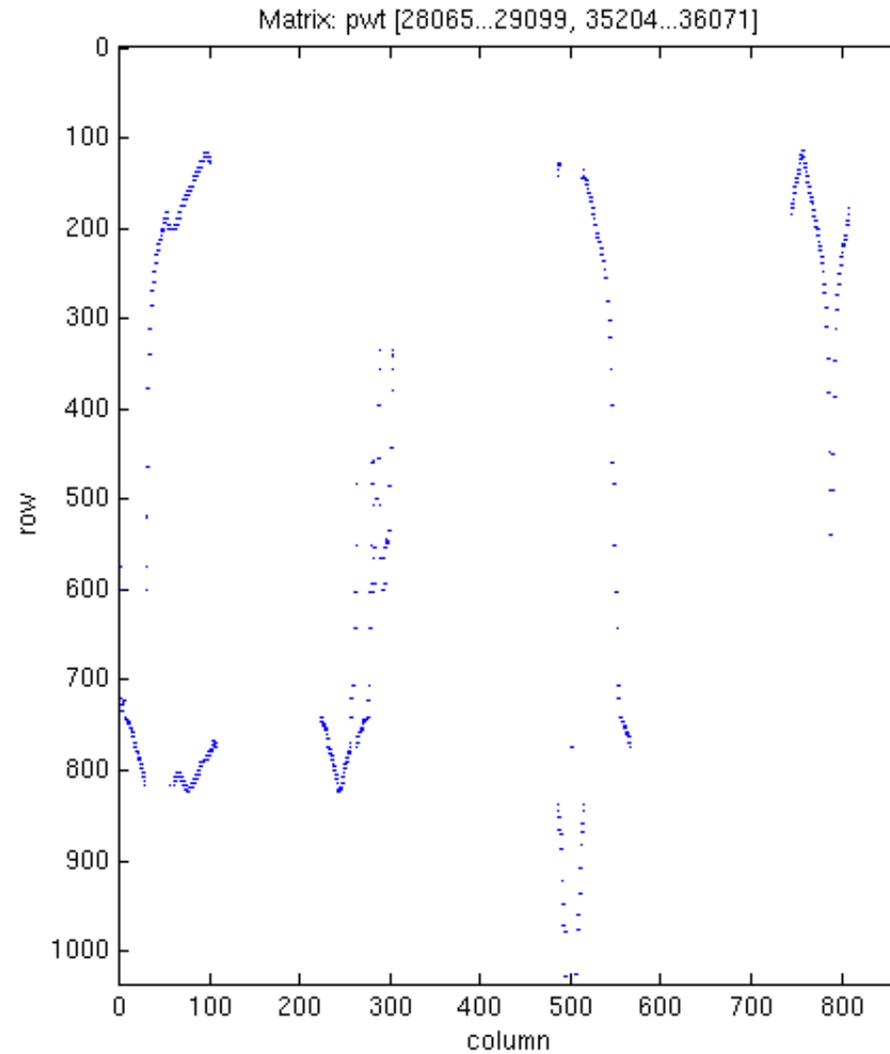
Matrix #22 (cont'd)- bayer02 (chemical process simulation)



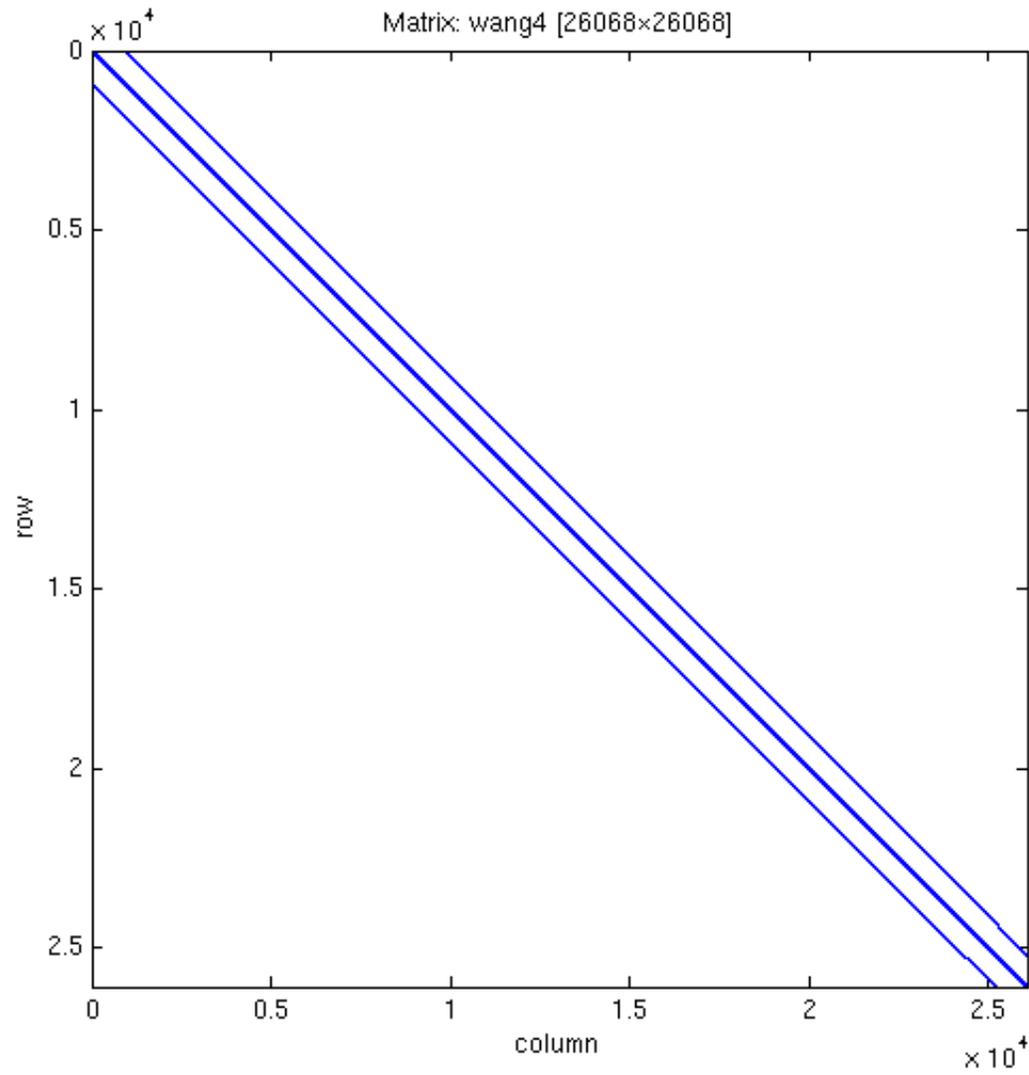
Matrix #27 - pwt (structural engineering)



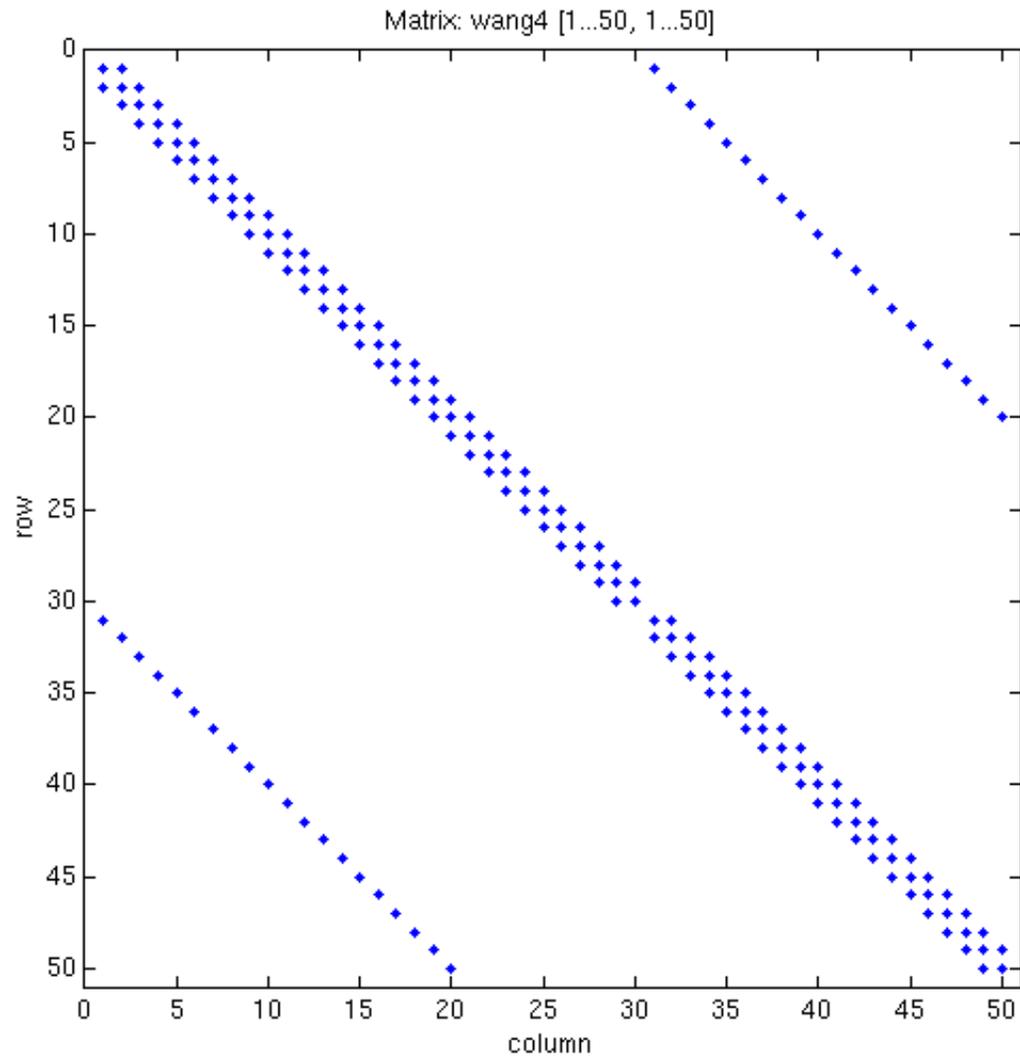
Matrix #27 (cont'd)- pwt (structural engineering)



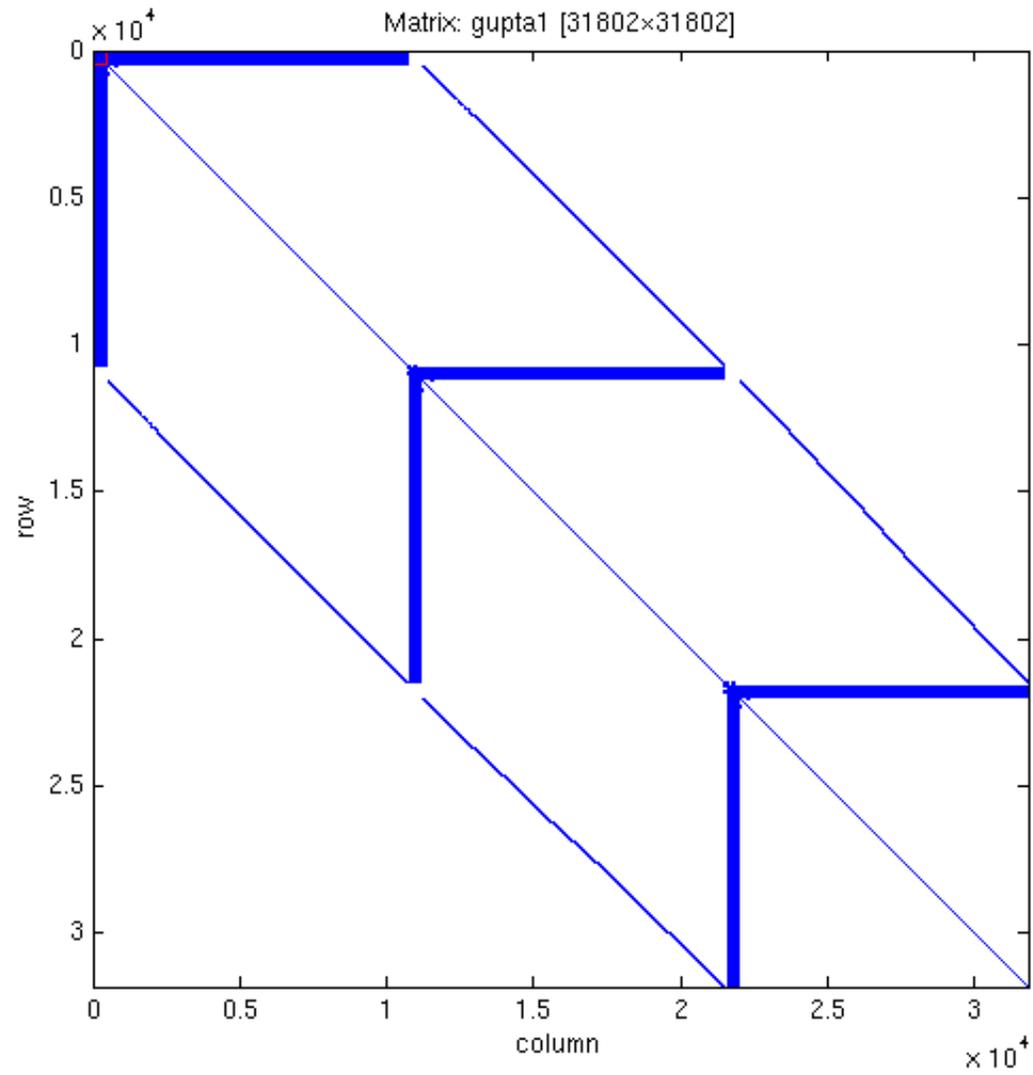
Matrix #29 - wang4 (semiconductor device simulation)



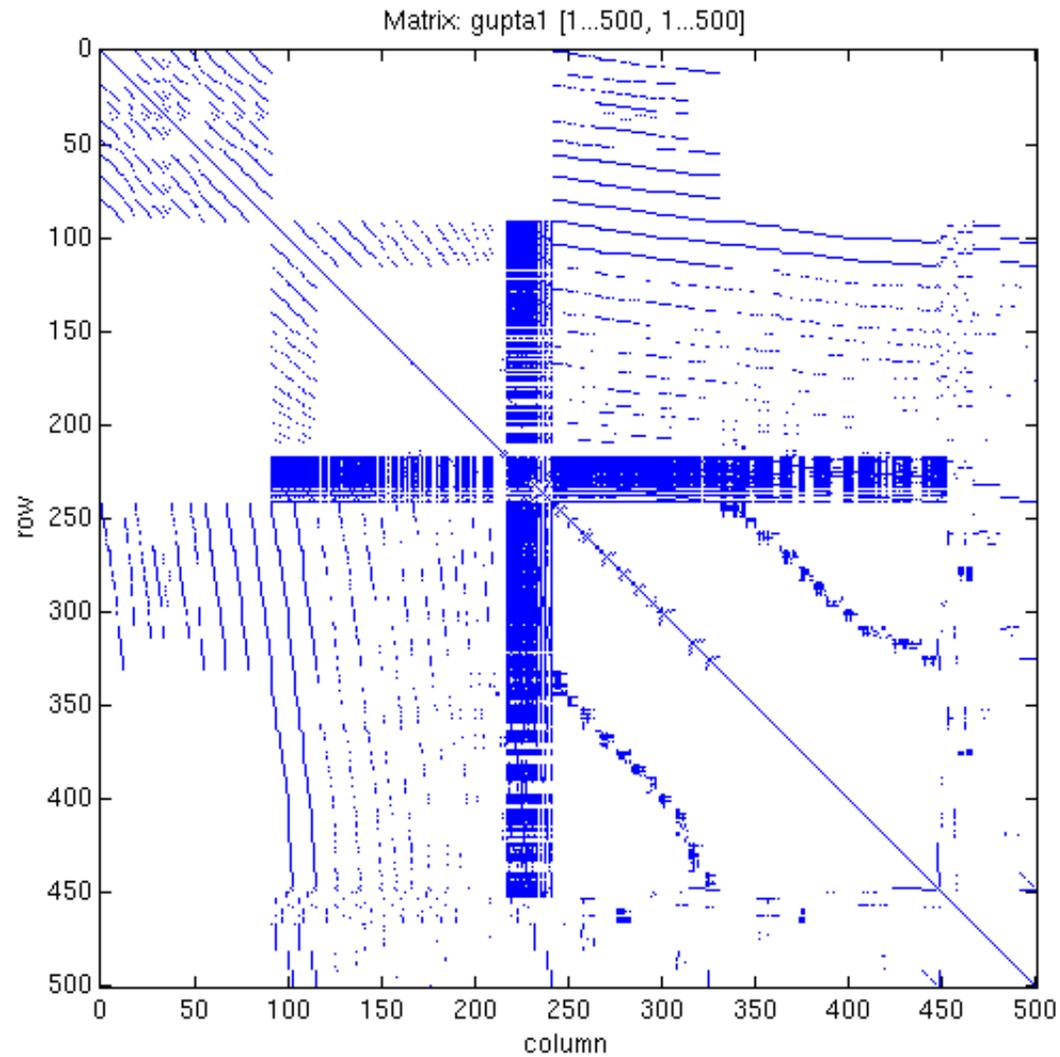
Matrix #29 (cont'd)-wang4 (semiconductor device sim.)



Matrix #40 - gupta1 (linear programming)



Matrix #40 (cont'd) - gupta1 (linear programming)



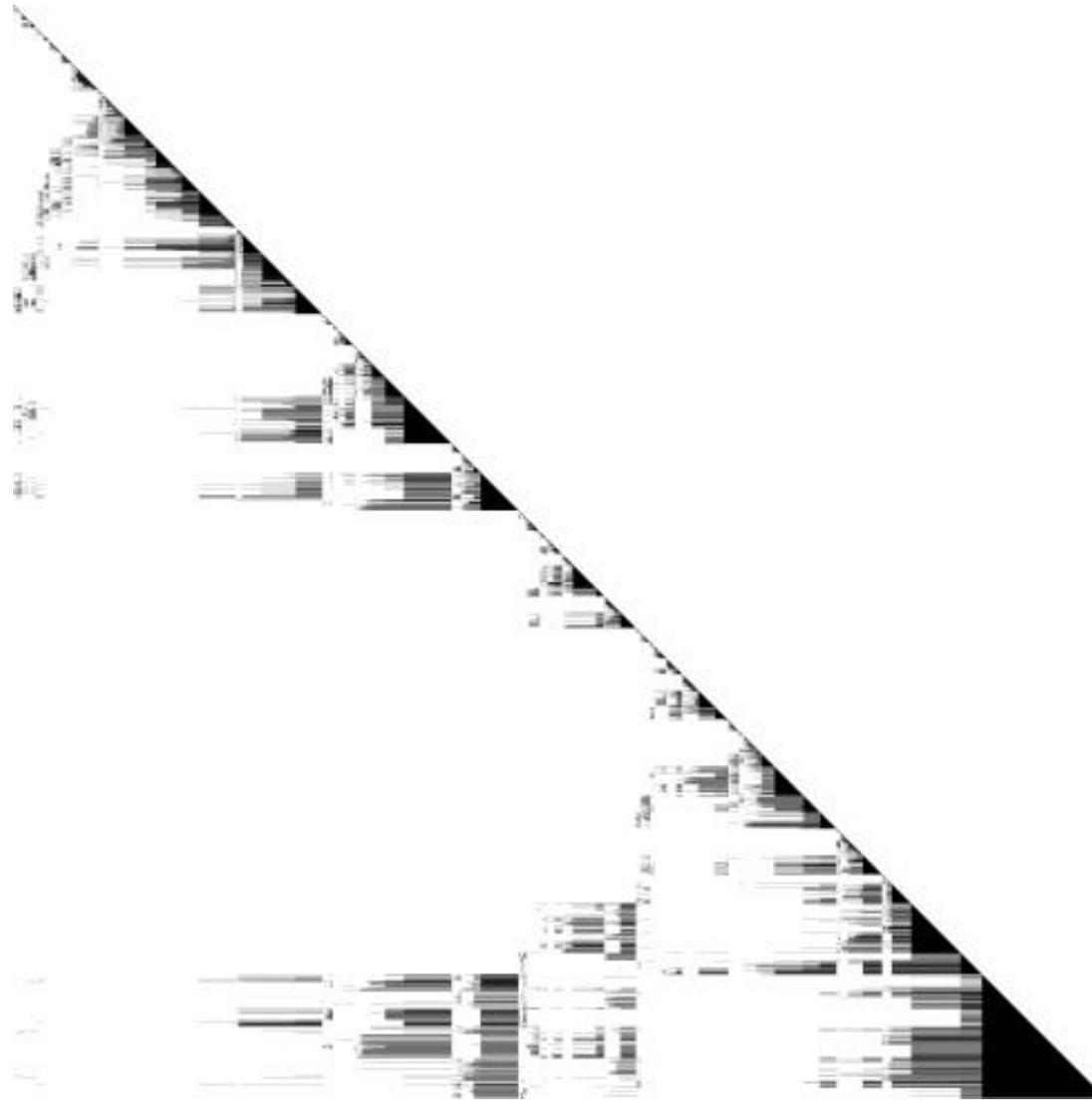
Symmetric Matrix Benchmark Suite

#	Matrix Name	Problem Domain	Dimension	No. Non-zeros
1	dense	Dense matrix	1,000	1.00 M
2	bcsstk35	Stiff matrix automobile frame	30,237	1.45 M
3	crystk02	FEM crystal free vibration	13,965	969 k
4	crystk03	FEM crystal free vibration	24,696	1.75 M
5	nasasrb	Shuttle rocket booster	54,870	2.68 M
6	3dtube	3-D pressure tube	45,330	3.21 M
7	ct20stif	CT20 engine block	52,329	2.70 M
8	gearbox	Aircraft flap actuator	153,746	9.08 M
9	cf2	Pressure matrix	123,440	3.09 M
10	finan512	Financial portfolio optimization	74,752	596 k
11	pwt	Structural engineering	36,519	326 k
12	vibrobox	Vibroacoustic problem	12,328	343 k
13	gupta1	Linear programming	31,802	2.16 M

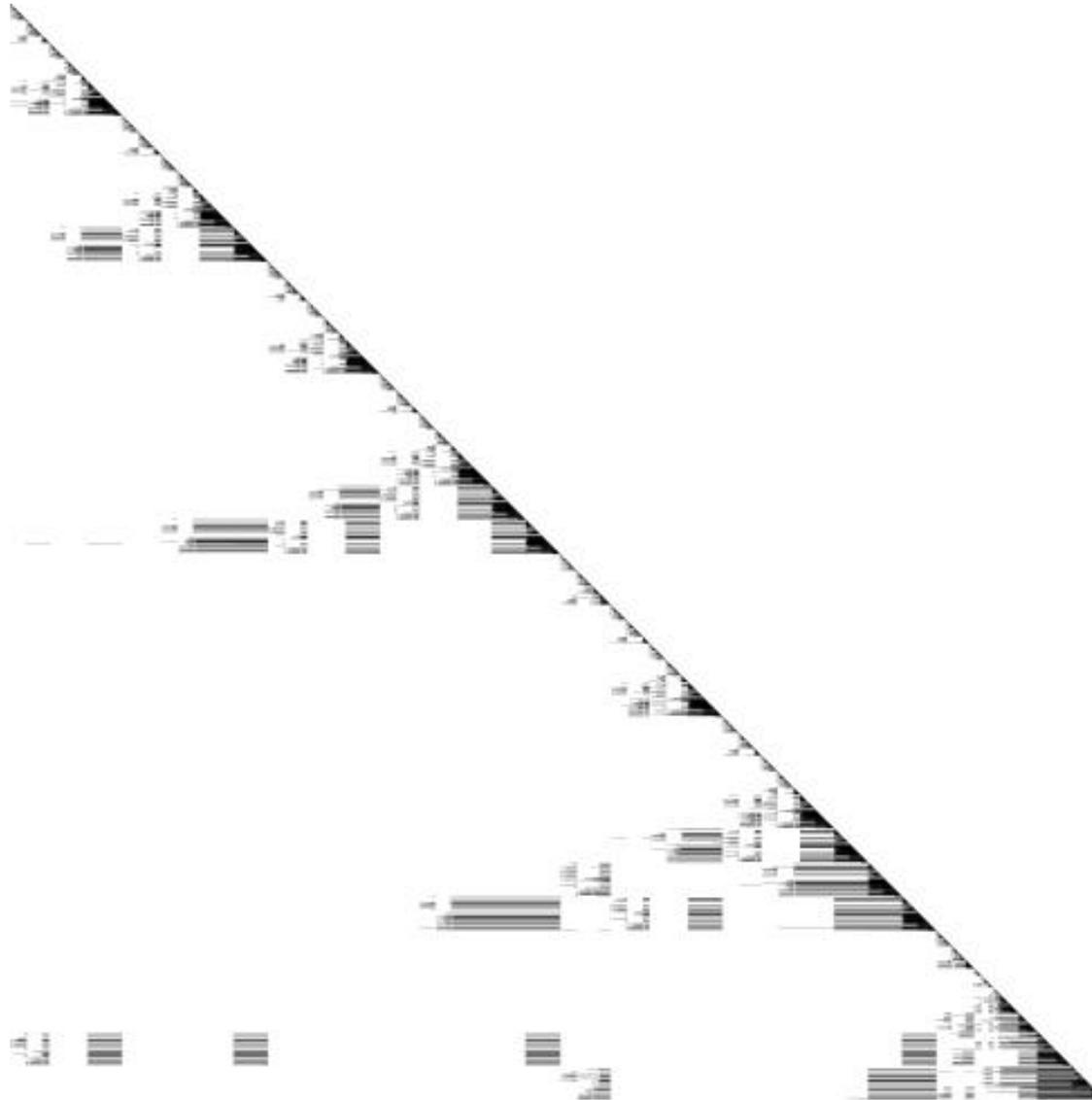
Lower Triangular Matrix Benchmark Suite

#	Matrix Name	Problem Domain	Dimension	No. of non-zeros
1	dense	Dense matrix	1,000	500 k
2	ex11	3-D Fluid Flow	16,214	9.8 M
3	goodwin	Fluid Mechanics, FEM	7,320	984 k
4	lhr10	Chemical process simulation	10,672	369 k
5	memplus	Memory circuit simulation	17,758	2.0 M
6	orani678	Finance	2,529	134 k
7	raefsky4	Structural modeling	19,779	12.6 M
8	wang4	Semiconductor device simulation, FEM	26,068	15.1 M

Lower triangular factor: Matrix #2 - ex11



Lower triangular factor: Matrix #3 - goodwin



Lower triangular factor: Matrix #4 - lhr10

